An Introduction: Neural Network Quantization and Parameterized Clipping Activation

Marc Thieme
Supervisor: Haibin Zhao

Karlsruher Institut für Technologie, 76131 Karlsruhe, Germany
usldt@student.kit.edu

Abstract. Quantization of neural networks is currently one of the most promising and well researched approaches to neural network compression and acceleration we have. It is a diverse research field tending in the direction of customizing and adapting ever more and creative parameters. The Parameterized Clipping Activation (PACT) quantization scheme was a contribution to this trend in that it opened up the clipping and quantized range of activations to be adapted during training.
In this paper, we give an overview over the quantization research field and detail the PACT algorithm as an example. The goal is to present materials in an approachable way and act as an introduction to the field with an in-depth example constituted by the PACT algorithm.

1 Introduction

Deep Neural Networks have revolutionized the fields of image, audio and video processing [14]. As phrased in [14], "Deep learning [has been] making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years". However, the size of deep neural networks has increased to the point where they can be not only hard to train, but also hard to deploy, specifically concerning their memory footprint, execution speed and training requirements. [28, 13] This hinders their use in resource-constrained environments, such as "electronic devices and services, from smartphones, smart glasses and home appliances, to drones, robots and self-driving cars" [19]. Extensive research is directed towards studying several approaches to mitigate this problem [6]. Examples consist of "channel pruning, [which] directly reduces feature map width [...] [and] shrinks a network into a thinner one" [12]. In Channel Pruning, redundant channels are eliminated by using sparsity-constraints, selecting channels to remove in a way which minimizes the error [33]. Similarily, network sparsification "attempts to reduce the number of operands [neurons and weights]", e.g. making a "heuristic search of unimportant elements" in the network [7]. Other approaches for model compression are "Knowledge distillation [, which] is a popular technique for training a small student network to emulate a larger teacher model" [25], neural architecture search as "the process of automating architecture engineering" [9], or lastoly dedicated architectures such as SqueezeNet [13].

1.1 Introduction to Neural Network Quantization

One promising and well-researched approach to network compression is quantization of neural networks [28]. Here, the overarching idea is to replace computations using 32-bit floating point numbers with computations using numbers of smaller bit-width, usually in a different representation. This reduces the memory footprint and the computational cost of the network [28]. Conceptionally, the "Quantization of weights is equivalent to discretizing the hypothesis space [the mappings from the input space to the output space [2]] of the loss function with respect to the weight variables" [6]. Therefore, the approach trades "accuracy for lower precision to achieve smaller models" [28], i.e. it trades computational efficiency for the effectiveness (and applicability) of the gradient descent during backpropagation. There exist numerous approaches to quantization largely differing in bit-widths they require and number representations they use. Inherently, they exhibit different capabilities in compression and inference speed increases on one hand and inference accuracy on the other hand. Therefore, the goal for research in the field is generally minimizing the accuracy loss that comes from quantizing and pushing the field to make those approaches enabling the largest compression potential and computational efficiency viable [28].

1.2 Overview of the Pact Algorithm

In works previous to [6], it was difficult incorporating the Linear Rectified Unit (ReLU) activation function (see section 2.2) effectively in quantization. The problem was that when "the output of the ReLU function is unbounded, the quantization after ReLU requires a high dynamic range" [6] and when the output was clipped to become unbounded, approaches would "not fully utilize the strength of back-propagation to optimally learn the clipping level because all the quantization parameters are determined offline and remain fixed throughout the training process" [6][3]. Other research did not use ReLU for the activation function in favour of leaving the threshold function unspecified but requiring its output range in [0; 1), which enables straightforward quantization [32]. However ReLU has proven very effective in Convolutional Neural Networks (CNN) (see section 2.1) because it leaks the gradient in the positive range through to previous layers during back propagation [21].

In order to preserve the effectiveness of ReLU, the PACT algorithm introduces a similar activation function which clips the output based on a parameter learned during training. As such, it introduces a clipping range dynamically per layer, which is meant to prevail the benefits of ReLU, while enabling effective quantization. More specifically, it identifies the need for varying quantization scales in ReLU due to large variations in the activation amplitude and consequently large variations in the optimal quantization ranges across the network [6].

1.3 Objectives and Audience

This paper gives an overview over quantization ideas and approaches. Afterwards, we first introduce concepts relevant to the PACT algorithm as presented in [6] and subsequently explain the algorithm in a more elaborate and approachable way than the original paper. It is intended for readers with a basic understanding of neural networks but no previous experience with quantization. The paper is structured as follows: Section 2 provides an overview of quantization and branches out to techniques neighbouring those applied in [6]. Section 3 introduces the concepts relevant for understanding and motivates the PACT algorithm. Section 4 provides a detailed explanation of the algorithm. Section 5 discusses extensions and variations of the algorithm published since its release.

2 Background of Neural Network Quantization

The following section constitutes a recollection of basic concepts and terminology. It is intended to be a primer for readers already familiar with the concepts. It is not intended to contain comprehensive explanations of the topics.

2.1 Overview of Convolutional Neural Networks

A neural network is typically organized in layers, with each layer containing a set of nodes that process information from the previous layer. When executed, the first layer (input layer) of the network assumes the values of the input of the network while the values of the last layer (output layer) are interpreted as the output. Neural networks are trained to minimize a loss function with respect to the actual and the desired output of the network. During so-called backpropagation and gradient descent, the gradient of the network's parameters with respect to the loss function is calculated and then used to update the network's parameters in the direction of minimal loss. By repeating this algorithm, ideally, the network finds a minimum of the loss function which corresponds to a satisfying predication performance of the network [23].

Convolutional neural networks (CNNs) are a type of neural network successfully used in image recognition tasks [11]. A CNN contains convolutional layers which convolve a set of learnable kernels across each channel of their input. Ideally after training, each kernel detects specific features such as edges, corners, or textures of the input image. By sequencing multiple convolutional layers, CNNs can learn complex and abstract features [31].

In a fully connected layer, all n nodes of one are connected to all m nodes of the previous layer. The output is calculated according to a weight matrix $W \in \mathbb{R}^{m \times n}$ and a bias vector $b \in \mathbb{R}^n$. Both are parameters of the network. The output is then given by $Wx + b$, where x is the vector of activations of the nodes in the previous layer. The output is then fed through an activation function introducing non-linear properties to the networks transformations, allowing it to learn complex data patterns [24]. For the activation functions, we will only discuss ReLU in this article.

2.2 Loss function and Gradient descent

The loss function in Neural networks is used as a quality measure for the predictions a neural network makes. It takes the networks' and the desired predictions of a network as inputs and calculates a measure for their disparity. By offering a quantifiable measure of how closely the network matches the desired predictions, it presents an objective whose minimization by gradient descent represents aligning the network's transformations to the desired ones [27].

Backpropagation with gradient descent is the process of adjusting the models weights to minimize the loss function. It calculates the gradient of the loss function with respect to each parameter by recursively calculating previous layers' based on the gradients calculated for successive layers. Along the way, it adjusts the parameters to decrease the loss function based on the calculated gradient [24].

ReLU Activation Function ReLU is an activation function which is used in neural networks. It is defined as

$$h(x) = max(0, x) \tag{1}$$

[21] and performs particularly well because it allows the gradient to propagate through layers of the neural network [11]. [6] use a variant of ReLU which is bounded by a parameter α learned during training. As such, their contribution can be boiled down to enabling ReLU activation function to be used effectively in a quantization context.

2.3 Introduction to Quantization Techniques

The representation of numbers within computers provides the basis for understanding quantization. Often, neural networks "use 32-bit floating point data types to represent their parameters as well as all of the computations involved with inference, meaning they require expensive floating point units to run" [28]. When applied to neural networks, quantization encompasses the reduction of these numerical representations' precision, shifting from n-bit precision to smaller m-bit precision. This transition can lead to a notable reduction in memory usage and computational requirements, a critical consideration for implementing neural networks on resource-limited devices [28, 19].

Different numerical representations can be used during quantization:

1. Floating Point Representation: Numbers are represented using scientific notation, e.g. according to IEEE floating point standard using 32 bits. They provide a broad dynamic range and higher precision for values closer to 0. However, multiplication is more complicated and costly than alternatives. [28].

2. Fixed Point Representation: Numbers are represented with a fixed number of digits before and after the decimal point. Multiplying two numbers is efficient because we can first multiply the representations as if they were integers and then shift the bits to align the decimal point [28].
3. Dynamic Fixed Point Representation: "Dynamic fixed point attempts to meet floating point and fixed point in the middle. [...]. Dynamic fixed point employs several scaling factors that are shared among a group of values, and these scaling factors are dynamically updated as the statistics of the group changes" [28].
4. Integer Representation: Only integer values are represented, providing straightforward calculations but naturally only integer precision, which can hamper accuracy [28].
5. Binary and Ternary Representation: Numbers are represented using only two (0 and 1) or three (-1, 0, and 1) values respectively. This yields very good compression ratio and enables hardware optimizations/simplifications [28, 16].

Additionally, the process of quantization can adopt either a uniform or non-uniform approach and can be symmetric or asymmetric. Uniform quantization maintains equal intervals between each quantization level, while non-uniform quantization permits variable intervals [28, 19, 15]. In symmetric quantization, 0 is quantized to zero. For asymmetric, also called affine quantization, this restriction is lifted [19].

[6] use uniform and symmetric quantization. It is prefaced with a clipping function (resembling ReLU) and quantizes from the clipping range $[0, \alpha]$ in turn to the quantized range $[0, \alpha]$ and use fixed-point representation with constant k bits precision.

2.4 Neural Network Quantization: Goals, Benefits and Limitations

When applying quantization to a neural network, the goal is generally to empower running on resource-constrained environments [28]. The problem without network compression often lies in network size, inference latency and energy consumption [28]. Also, quantization can enable hardware optimizations for some specific number representations which would not be possible conventionally [32, 16].

By network size or memory footprint of a network, we refer to the amount of data required to store the model parameters. As models can have millions of parameters filling about 100MB [5] their deployment can be a challenge on devices with restricted memory capabilities, e.g. on the edge. Also, a large memory footprint can me a major hindrance in applications where the model is transmitted across a network, e.g. image recognition models in cars [13]. Quantization addresses this problem by reducing the size of the model parameters thereby decreasing the overall memory footprint of the entire model. [28]

As quantization reduces bit-widths and ideally simplifies the number representation, operating on them naturally becomes cheaper. Also, the quantized representations can enable special optimizations, e.g. for binary quantization, dot product operations can be implemented using only bitshifts [32] or for "ternary weights[,] one can get rid of the multipliers and use adders instead" [16]. For the same reason, inference on the quantized networks save energy [28]. For instance, an adder component on a circuit board for 32-bit floating point numbers consumes 30 times more energy than an 8-bit adder component [5].

2.5 Limitations and Challenges in Conventional Quantization Techniques

Quantization techniques, while beneficial in minimizing the memory footprint and computational costs of neural networks, also present their own set of challenges. The most prominent among these are accuracy loss and a discrete search space which needs to be worked around and generally results in slower converges during training.

At the time of writing this, quantizing a network can still lead to 20% to 30% decrease in accuracy even for the best choices in the quantization scheme [28]. Usually, using fewer bits sacrifices accuracy. This trade-off between model accuracy and computational efficiency is a fundamental challenge in neural network quantization [28].

When quantizing a neural network, the parameters are mapped to a discrete set of values. As a result, the gradient becomes trivial almost everywhere and the traditional gradient descent would not progress. To combat this, alternative techniques to the analytical gradient descent are introduced, often leading to more complex optimization landscapes (e.g. the search space grows exponentially with the number of layers when using mixed precision quantization [8]). This makes the process of finding a good local minimum more challenging and ultimately results in slower convergence and lies at the root of the accuracy degradation [28].

2.6 Overview of existing Neural Network Quantization Techniques

Neural network quantization methodologies primarily categorize into two types: Quantization Aware Training (QAT) and Post-Training Quantization (PTQ). QAT integrates quantization into the training phase, thereby enabling the network to adapt to the quantization errors during training. On the contrary, PTQ applies quantization after the training process has been completed. This method is more effective during training since the search space remains continuous, albeit the sudden introduction of quantization can lead to significant performance loss, as the model has not been equipped to adjust to the quantization errors during the training phase [28].

Both QAT and PTQ employ a range of quantization strategies that largely match the general quantization techniques discussed in section 2.3. Uniform

quantization refers to evenly spaced quantization levels, while non-uniform quantization adjusts the levels based on the distribution of weights, potentially providing higher precision only where it is needed. However, this non-uniformity makes operations more complex as it generally carries along an additional normalization step before numbers can be operated together [15]. Another promising approach is mixed precision quantization [28]. It varies and determines custom bit-widths where needed: With mixed precision quantization, "each layer has a tailored bit width and precision because mixed precision recognizes that some layers may benefit from more bits whereas others can afford to use fewer bits" [28]. [8] demonstrate one example for employing mixed precision quantization effectively.

As the search space becomes discrete when using quantization, the standard gradient descent cannot be employed meaningfully. Two solutions for this problem are introducing a proxy value for the analytical derivation. These are the popular [29] Straight-Through-Estimator (STE) or the backpropagation approach in Monte-Carlo-Quantization (MCQ). The former approximates the discrete quantization function with a similar differentiable function and delegates the gradient of the former to the gradient of the latter [1, 29]. In contrast, Monte Carlo Quantization (MCQ) introduces a stochastic component to the gradient estimation process, approximating the gradient using the Monte Carlo method [17].

3 Overview over the PACT Algorithm

3.1 Summary

The PACT algorithm consists of an activation function $PACT(\cdot)$ based on ReLU and the subsequent quantization operator. This operator is similar to $\mathbf{quantize}_k$ proposed by [32] but scales the quantization range by a parameter α. This parameter distinguishes the approach from other quantization schemes [32, 16] since it allows dynamically adapting the quantization scale to the problem domain and the locale in the neural network [6]. α is a parameter of the search space and is optimized during training.

In this section, we will first introduce prerequisite knowledge which is not original in [6]. Afterwards, we will present the algorithm in detail and discuss nuances and empirical recommendations presented by the authors.

3.2 Technical Concepts used in the Pact Algorithm

Regularization Regularization is a technique used to prevent overfitting [30]. Overfitting, in turn, is a "fundamental issue in supervised machine learning which prevents us from perfectly generalizing the models to well fit observed data on training data [sic], as well as unseen data on testing sets". The problem entails that "the model performs perfectly on [the] training set, while fitting poorly on testing sets" [30]. Generally, this phenomenon is caused either by the

model learning noise in the training data or by the model's parameter set being complex enough to fit the training data so accurately that it does not generalize meaningfully to data not in the training set [30].

Generally, the goal of regularization is to "minimize the weights of the features which have little influence on the final classification" [30]. In practice, this is accomplished by augmenting the loss function by an additional term which penalizes large weights. For L1 and L2 regularization respectively, the regularized loss function $\widetilde{c}(\omega, X, y)$ becomes

$$\textbf{L1:} \qquad \widetilde{c}(\omega, X, y) = c(\omega, X, y) + \lambda \sum_{i=1}^{n} |\omega_i| \tag{2}$$

$$\textbf{L2:} \qquad \widetilde{c}(\omega, X, y) = c(\omega, X, y) + \underbrace{\lambda \sum_{i=1}^{n} \omega_i^2}_{\text{regularization term}} \tag{3}$$

where $\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}$ are the parameters of the network, X is the training set, y is the labelled data and λ is the regularization coefficient dictating the strength of the regularization effect. Since the parameter updates during backpropagation always follow the gradient of the loss function in the decreasing direction, the regularization term "provides incentive" for the parameters not to grow extremely large. This can prevent features from being overrepresented in the model due to the parameters of the feature contributing excessively to the output of the model [30].

Backpropagation on discrete functions: Straight Through Estimator As mentioned in section 2 the gradient of a discrete function is everywhere either zero or undefined [29, 19]. This is a problem in neural network quantization [1] because the functions used to quantize the weights, activations or gradients is discrete by definition.

However, the gradient is necessary to update the parameters (e.g. the weights) of the neural network during training. In particular, if the incoming gradient for a neuron or a convolution kernel is zero, the parameters would not change no matter the learning rate. Ideally, you would want the parameters to move closer in the direction whose next quantization level decreases the loss function. PACT uses the Straight Through Estimator (STE) to approximate this judgement [6], which, for uniform quantization, generally approximates the derivative of the quantization with the derivative of the identity function [1, 29, 19].

The Quantize Operator [32] present an operator which implements a quantization function for the forward pass and an STE for the backward pass.

This operator is similar to the following operator, which [6] append to their custom activation function. Adhering to their notation, [32] define the $\mathbf{quantize}_k$ operator as follows:

$$\textbf{FORWARD:} \qquad x_q = \frac{1}{2^k - 1} round\left(x \cdot \left(2^k - 1\right)\right) \tag{4}$$

$$\textbf{BACKWARD:} \qquad \frac{\partial c}{\partial x} = \frac{\partial c}{\partial x_q} \tag{5}$$

where x is the input to the quantization function, x_q is the quantized output, k is the number of bits used to represent the quantized value and $c(\cdot)$ is the loss function.

We first observe that, during the forward pass, this operator implements a symmetric uniform quantization operation over the clipped and also quantized range $x, x_q \in [0, 1]$. Second, we observe the application of an STE for the backward pass. By rewriting the equation to

$$\frac{\partial c}{\partial x_q}\frac{\partial x_q}{\partial x} = \frac{\partial c}{\partial x} = \frac{\partial c}{\partial x_q} \tag{6}$$

and deriving:

$$\frac{\partial x_q}{\partial x} = 1 \tag{7}$$

we see the resemblance of the STE in its canonical form for uniform quantization [29].

Having defined the behavior of $\mathbf{quantize}_k$ for the backward pass, we can use it in training a model. However, the quantization range is fixed to $[0, 1]$ and the quantization step size is fixed to $\frac{1}{2^k-1}$. Consequently, it assumes a bounded activation function $h(\cdot)$ which ensures activations fall in this range [32]. This is a problem because the quantization range and step size are not adapted to the problem domain. For example, if the input to the quantization function is always in the range $[-0.5, 0.5]$, the quantization range $[-1, 1]$ is unnecessarily large. This results in an unnecessary loss of precision. When ReLU is used, the range of activation is $[0, \text{inf}]$ [6]. Attempting to use this as the quantization range for a uniform quantization yields an infinitely large step size, making the operation impossible in practice. This problem will be solved by the PACT activation function, which uses a threshold function akin to ReLU but bounded by a parameter α which is learned during training.

3.3 Principles and Motivation of the Pact Algorithm

Those are the techniques and topics relevant to the PACT algorithm. In the following section, they will be used to construct a quantization scheme which enables the use of an activation function akin to ReLU and enables a model during training to adjust the quantization scale used in quantizing the activations according to a specific model's and problem domain's requirements.

4 Algorithm Details

4.1 The PACT Activation Function

As stressed in the previous section, the PACT algorithm is based on the idea of encorparating ReLU into a framework which quantizes activations. [6] propose the PACT activation function to serve this purpose. They define $PACT\colon \mathbb{R} \to [0, \alpha]$ as

$$\mathrm{PACT}(x) = \begin{cases} 0 & \text{if } x \in (-\inf, 0) \\ x & \text{if } x \in [0, \alpha) \\ \alpha & \text{if } x \in [\alpha, \inf) \end{cases} \tag{8}$$

where $\alpha \in (0, \inf)$ is a learnable parameter. Note that the PACT function is equal to ReLU appended to another clipping function whose clipping range is $[0, \alpha]$.

As previously stated, the idea is to leverage the training capabilities of the model to adjust α in such a way that the clipping range is the value range which carries the most "meaning" and does not disturb the inference capabilities of the network [6].

4.2 The PACT Function in the Backward Pass

Before covering the quantization-aspect of the PACT algorithm, we first look at the PACT function itself in more detail.

While its use in the forward pass is expressed by its definition, we need to derive its gradient in order to use it in the backward pass. Without quantization, using the PACT function in the backward pass is trivial. Note that we need the gradient of the loss function with respect to x but that we can calculate this gradient $\frac{\partial c}{\partial x} = \frac{\partial c}{\partial PACT(x)} \cdot \frac{\partial PACT(x)}{\partial x}$ by using the chain rule and the derivative of $PACT(x)$ with respect to x.

Whilst not being differentiable in $x = 0$ and $x = \alpha$, we can calculate the derivative of PACT based on its piecewise derivatives.

$$\frac{\partial \mathrm{PACT}(x)}{\partial x} = \begin{cases} 0 & \text{if } x \in (-\inf, 0) \\ 1 & \text{if } x \in [0, \alpha) \\ 0 & \text{if } x \in [\alpha, \inf) \end{cases} \tag{9}$$

As such, the derivative of PACT behaves akin to the derivative of ReLU for $x \in (-\inf, \alpha]$, only for $x \geq \alpha$ the derivative is zero instead of one. Note that ReLU's supposed property of being able to pass the gradient through to previous layers is conserved for $x \in (-\inf, \alpha)$. Note further that conceptually, this range $(-\inf, \alpha)$ is meant to be the "relevant" value range for values passed into the function so ideally, the deviation from ReLU in the range $x \geq \alpha$ does not hamper the models training and inference capabilities significantly.

4.3 Training the parameter α

The parameter α is trained in the same way the other network parameters are learned by minimizing the loss function and by using gradient descent. This implies that we need to be able to calculate the derivative $\frac{\partial c}{\partial \alpha}$ of the loss function with respect to α. Recap that after obtaining this derivative, we can adjust α in the opposite direction to decrease the loss function further. Also recap that the derivative of the PACT function with respect to α is enough to infer the derivative of the loss function according to the chain rule. Without quantization, this derivative is calculated similarly to the gradient with respect to the input vector x above [22]:

$$\frac{\partial \text{PCACT}}{\partial \alpha}(x) = \begin{cases} 0 & \text{if } x \in (-\infty, 0) \\ 1 & \text{if } x \in [0, \alpha) \\ 0 & \text{if } x \in [\alpha, \infty) \end{cases} \tag{10}$$

When writing the PACT and the ReLU function as

$$\textbf{FORWARD:} \qquad \text{ReLU}(x) = \max(0, x) \tag{11}$$

$$\textbf{BACKWARD:} \qquad \frac{\partial \text{ReLU}}{\partial x}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{12}$$

$$\textbf{FORWARD:} \qquad \text{PACT}(x) = \max(0, \min(\alpha, x)) \tag{13}$$

$$\textbf{BACKWARD:} \qquad \frac{\partial \text{PACT}}{\partial x}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \text{ and } x \leq \alpha \\ 0 & \text{if } x > \alpha \end{cases} \tag{14}$$

their resemblance becomes apparent.

As we can see, the approach and calculations for incorporating the PACT function into a model without quantization is trivial. Following, we will look at how [6] encorparates this quantization into their algorithm.

4.4 Quantizing after PACT

[6] contribution is an improved approach to quantizing activations in CNNs. In the last section, we looked at the general activation function PACT. In this

section, we will look at the quantization mechanism they combine with this activation function.

PACT assumes a constant bit-width k which is used for the fixed-point number representation in their model [6].

Let y be the activation after applying the PACT function. Let y_q be the output after quantization, then y_q is calculated using

$$y_q = \text{round}(y \cdot \frac{2^k - 1}{\alpha}) \cdot \frac{\alpha}{2^k - 1} \tag{15}$$

Therefore, the activation y is quantized to the nearest value in the range $[0, \alpha]$ which is representable using k bits. The quantization is done by first scaling the input y to the range $[0, 2^k - 1]$ and then rounding it to the nearest integer. The result is then scaled back to the range $[0, \alpha]$. As discussed previously, this represents a uniform and symmetric quantization akin to the $\mathbf{quantize}_k$ operator in [32]. The difference is that $\mathbf{quantize}_k$ quantizes values $x \in [0, 1]$ while the above operation quantizes values in the range $x \in [0, \alpha]$ [6, 32]. As this range depends on the trained parameter α, the same bit vector b almost always encodes different real numbers in different training epochs because α changes.

4.5 Analyzing the quantization

Briefly, we want to demonstrate how to interpret the quantized values. Ideally, we could assign each quantization level to one possible configuration of our k-bit number. E.g, if $(b_1, \ldots, b_k) \in \{0, 1\}^k$ is an arbitrary bit-vector, we would match the quantized value $y_q = \frac{\alpha}{2^k-1} \cdot \sum_{i=1}^{k} \left(2^{i-1} \cdot b_i\right) \in \{0, \frac{\alpha}{2^k-1}, \frac{2\alpha}{2^k-1}, \frac{3\alpha}{2^k-1} \cdots, \alpha\}$ to it. Note the scaling factor $\frac{\alpha}{2^k-1}$ to the integer value.

[6] mention they use fixed-point multiply-accumulate (MAC) [19] for their experiments. However, fixed-point numbers require the scaling factor to be a power of 2 [28]. However, the scaling factor $\frac{\alpha}{2^k-1}$ can be arbitrary. It remains unclear to us how the authors implemented their calculations exactly.

They do, however, recommend to share the value of α per layer. Amongst yielding good accuracy [6] and working well, "this choice also reduces hardware complexity because α needs to be multiplied only once after all MAC operations in reduced-precision in a layer are completed" [6]. Also, they recommend to initialize α larger than expected as reducing α in combination with a regularizing it worked better than the other way around as per the observations by the authors. [6].

4.6 Using the PACT Function in Backward Pass

So far, we have discussed the application of the PACT function in the forward pass, specifically in combination with quantization. We have also discussed its behavior in the backward pass when values are not quantized. Lastly, we look at their combination in the context of the backward pass and backpropagation. Our objective is to compute the derivative of the loss function. Using the chain

rule, the derivative of the PACT function itself with respect to α suffices for that.

[6] use an STE for that. Similar to its application for the $\mathbf{quantize}_k$ operator in [32], the STE approximates the PACT function as the uniform distribution and approximates the derivative to be equal to the derivative of the identity function. As such, we start with the transformation $\frac{\partial y_q}{\partial \alpha} = \frac{\partial y_q}{\partial y}\frac{\partial y}{\partial \alpha}$ using the chain rule, we approximate $\frac{\partial y_q}{\partial y} \approx 1$ according to the STE and we plug in our result from equation 10 to obtain

$$\frac{\partial y_q}{\partial \alpha} \approx \frac{\partial y}{\partial \alpha} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{16}$$

which gives the α component of the gradient and can be used in backpropagation.

4.7 Regularization

[6] also propose a regularization term for α to be added to the loss function. They use L2 regularization. Consequently, the term added per parameter (when using one value for α per layer, this means one additional regularization term per layer) equates to

$$\widetilde{c}(\omega, X, y) = c(\omega, X, y) + \lambda_\alpha \cdot \alpha^2 \tag{17}$$

or for the whole model

$$\widetilde{c}(\Omega, X, y) = c(\Omega, X, y) + \sum_{i=1}^{n} \left(\lambda_{\alpha_i} \cdot \alpha_i^2\right) \tag{18}$$

The authors' idea behind this regularizer is to depress large values of α which equate to large clipping ranges and therefore low precision in these ranges [6].

5 Applications and Future Directions

The neural network quantization field is being actively researched [28]. As the PACT algorithm was published in 2018, novel ideas have come up since surpassing PACT in terms of accuracy [8, 4, 26], hardware utilization [4], or proposing novel complementary [10] and enabling promising results with alternative approaches such as PTQ [18, 20]. In this section, we will discuss two of these algorithms to give examples of developments since PACT.

5.1 Extensions and Variations of the Pact Algorithm

Two examples for developments succeeding the PACT algorithm will be highlighted, both employing alternative quantization mechanisms. The former will be using mixed-precision quantization, intelligently varying the bit-width used for each layer. The latter is employing non-uniform quantization.

HAWQ The Hessian Aware Quantization (HAWQ) algorithm uses mixed-precision quantization, seeking to pinpoint those layers volatile to perturbation and quantization noise. They employ higher bit-widths and precision for those layers they deem to be more volatile based on the second-order derivative of the loss function. Those layers where this measure is highest are estimated to be more volatile than others [8, 28]. They use "the Hessian matrix[,] (a matrix of the second derivatives), to determine how sensitive the weights and activations are. [...] The key observation is that layers with higher Hessian spectrum (larger eigenvalues) have a more volatile loss. These layers are prone to more fluctuations in the loss when even a small amount of quantization noise is introduced (e.g., by rounding errors). [...] The idea is that a flat loss magnifies noise, e.g., quantization noise, significantly less than a region with sharper curvature in their loss. Based on this Hessian information, they manually select the bit widths for each layer. Another key insight from HAWQ is that the order in which layers are quantized is important and affects accuracy loss. [...] They argue that [...] [t]he less sensitive layers adjust well to the introduction of quantization noise, so it is better to "lock in" the quantized values of the more sensitive layers first, allowing the less sensitive layers to recalibrate during this time." [28]

Non-Uniformity Generally, non-uniform quantization of neural networks is apt "to better fit underlying distributions and mitigate quantization errors [...] by adjusting the quantization resolution according to the density of real-valued distribution." However, as new projection steps are needed to operate on non-uniformly quantized numbers, the performance overhead becomes large enough that it is usually worth maintaining look-up-tables for the operations, which still incurs significantly worse performance than the fixed-point or integer multiplication variants. [15]

[28] propose a method "maintaining the [same] hardware projection simplicity as uniform quantization". In the process, they propose a variation on the STE which they call Generalized Straight Through Estimator (GSTE). It looks at each quantization level separately and fits a differentiable slope through the interval in a way such that "the influence of the threshold parameter α_1 to the network is decently encoded in the backward approximation function". α_1 refers to the length of the segment, and therefore dictates the end of the clipping range.

As for the quantization itself, they introduce the "nonuniform-to-uniform-quantizer (N2UQ) with its forward pass formulated as:

$$x^q = \begin{cases} 0, & x^r \leq T_1 \\ 1, & T_1 \leq x^r T_2 \\ \dots & \dots \\ 2^n - 1 & x^r \geq T_{2^n-1} \end{cases} \tag{19}$$

[15] where n is the number of bits, T represents learnable thresholds, and x^r and x^q are the real and quantized values, respectively." Using this

power-of-two quantizer [19] yielding fixed quantization levels in their design, they can achieve learnable non-uniformity through the adaptable segmentation, "while output[ting] uniformly quantized weights and activations to accommodate fast bitwise operations without the post-processing step between quantization and matrix multiplication." [15]

References

1. Bengio, Y., Léonard, N., Courville, A.: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation
2. Blockeel, H.: Hypothesis space
3. Cai, Z., He, X., Sun, J., Vasconcelos, N.: Deep Learning With Low Precision by Half-Wave Gaussian Quantization. pp. 5918–5926
4. Chang, S.E., Li, Y., Sun, M., Shi, R., So, H.K.H., Qian, X., Wang, Y., Lin, X.: Mix and Match: A Novel FPGA-Centric Deep Neural Network Quantization Framework. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 208–220
5. Chin, H.H., Tsay, R.S., Wu, H.I.:
6. Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.J., Srinivasan, V., Gopalakrishnan, K.: PACT: Parameterized Clipping Activation for Quantized Neural Networks
7. Deng, L., Li, G., Han, S., Shi, L., Xie, Y.: Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. Proceedings of the IEEE **108**(4), 485–532
8. Dong, Z., Yao, Z., Gholami, A., Mahoney, M., Keutzer, K.: HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 293–302. IEEE, Seoul, Korea (South)
9. Elsken, T., Metzen, J.H., Hutter, F.:
10. Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., Joulin, A.: Training with Quantization Noise for Extreme Model Compression
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. pp. 770–778
12. He, Y., Zhang, X., Sun, J.: Channel Pruning for Accelerating Very Deep Neural Networks. pp. 1389–1397
13. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size
14. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444
15. Liu, Z., Cheng, K.T., Huang, D., Xing, E., Shen, Z.: Nonuniform-to-Uniform Quantization: Towards Accurate Quantization via Generalized Straight-Through Estimation
16. Mishra, A., Nurvitadhi, E., Cook, J.J., Marr, D.: WRPN: Wide Reduced-Precision Networks
17. Mordido, G., Van Keirsbilck, M., Keller, A.: Monte Carlo Gradient Quantization. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 3087–3095. IEEE, Seattle, WA, USA
18. Nagel, M., Amjad, R.A., van Baalen, M., Louizos, C., Blankevoort, T.: Up or Down? Adaptive Rounding for Post-Training Quantization

19. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T.: A White Paper on Neural Network Quantization
20. Nahshan, Y., Chmiel, B., Baskin, C., Zheltonozhskii, E., Banner, R., Bronstein, A.M., Mendelson, A.: Loss Aware Post-training Quantization
21. Nair, V., Hinton, G.E.:
22. Park, E., Kim, D., Yoo, S., Vajda, P.: Precision Highway for Ultra Low-Precision Quantization
23. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117
24. Sharma, S., Sharma, S., Athaiya, A.: ACTIVATION FUNCTIONS IN NEURAL NETWORKS. International Journal of Engineering Applied Sciences and Technology **04**(12), 310–316
25. Stanton, S., Izmailov, P., Kirichenko, P., Alemi, A.A., Wilson, A.G.: Does Knowledge Distillation Really Work? In: Advances in Neural Information Processing Systems. vol. 34, pp. 6906–6919. Curran Associates, Inc.
26. Uhlich, S., Mauch, L., Cardinaux, F., Yoshiyama, K., Garcia, J.A., Tiedemann, S., Kemp, T., Nakamura, A.: Mixed Precision DNNs: All you need is a good parametrization
27. Wang, Q., Ma, Y., Zhao, K., Tian, Y.: A Comprehensive Survey of Loss Functions in Machine Learning. Annals of Data Science **9**(2), 187–212
28. Weng, O.: Neural Network Quantization for Efficient Inference: A Survey
29. Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., Xin, J.: Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets
30. Ying, X.: An Overview of Overfitting and its Solutions. Journal of Physics: Conference Series **1168**, 022022
31. Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks
32. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients
33. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware Channel Pruning for Deep Neural Networks. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc.