Quantization of Neural Networks: An Exploration of Techniques and Trade-offs

Tjaard Pfitzner

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

Abstract. With the recent advances in artificial intelligence, neural networks are becoming increasingly important. However, their widespread deployment on resource-constrained devices such as mobile phones and embedded systems is limited by their high memory and computational requirements. Neural network quantization allows for reducing the size and computational complexity of neural networks at the cost of precision and performance. This paper provides an exploration of the different techniques used for neural network quantization, with an emphasis on Quantization-Aware Training, a method that incorporates quantized parameters during the training process, where the network learns to accommodate the reduced precision and maintain performance. We will then look at Post-Training Quantization, which quantizes the network after the training process, therefore offering a fast and resource-saving method with the trade-off of lower accuracy.

Keywords: neural network, quantization-aware training

1 Introduction

Neural networks have been shown to be highly effective in various tasks, ranging from natural language processing [10] to computer vision [13]. This effectiveness, however, comes with a trade-off in complexity. According to [17], as neural networks increase in complexity, caused by the number of parameters, new challenges arise when it comes to deploying them on resource-constrained devices, such as mobile phones and embedded systems. The high memory and computational requirements for large neural networks limit their usability in domains, where efficiency and low power consumption are important.

A promising solution to address these challenges lies in neural network quantization. Part of neural network quantization is reducing the precision of weights and activations in a network from floating point values (typically 32-bit) to lower bit-width representation, such as 8-bit integer values. By the quantization of the network's parameters, a significant reduction in memory usage and computational complexity can be achieved [17].

Despite its benefits, quantization has a major downside. According to [20] and [14], it introduces a trade-off between complexity and accuracy. This is no surprise because the parameters are mapped to a lower bit-width representation during quantization and therefore have a lower precision, leading to a loss in model accuracy. This adaptation necessitates developing and using algorithms and techniques that can reduce the negative effects of quantization while making sure that the model retains the speed and size enhancements suggested by the overarching quantization concept [9].

In this paper, we will shine light on some of the techniques used to mitigate the negative effects of quantization. We will mainly emphasize the so-called Quantization-Aware Training, which involves training neural networks with an awareness of the quantization process, allowing the models to adapt and optimize their performance under reduced precision representations. In Section 2, we will give a brief overview of any prerequisite knowledge, while emphasizing neural networks. Section 3 will provide inside on the process of quantizing and dequantizing values to be used in the techniques discussed in Section 4, where we take a look at Quantization-Aware Training and compare it to another method used for neural network quantization called Post-Training Quantization. Finally, we will provide a general summary and conclusion.

2 Prerequisite Knowledge

In this section, we will 1) give an overview of neurons, 2) see how we can connect multiple neurons to form large neural networks and 3) explore how we use the back-propagation algorithm to train these neural networks.

2.1 Artificial Neurons

An artificial neuron is the smallest unit of a neural network [18]. Artificial neurons are modelled in a way that reassembles biological neurons. Multiple inputs get weighted and summed up in a linear combination. This combination is then passed through an activation function to calculate the final output. Figure 1 shows a graphical representation of a single neuron, indexed by j. Each input value x_i gets multiplied by its respective weight w_{ij}. All products are then summed up using the transfer function. Usually, we add a bias value b to the sum. The output of this linear combination is called the net input net_j. The net input gets passed through an activation function φ and if the value surpasses a certain threshold θ_j, then the neuron fires and we get an output value o_j. [23] gives a mathematical definition of the net input via:

$$net_j = \sum_{i=1}^{n} x_i \cdot w_{ij} + b.$$

The output of the neuron is defined as:

$$o_j = \varphi(net_j).$$

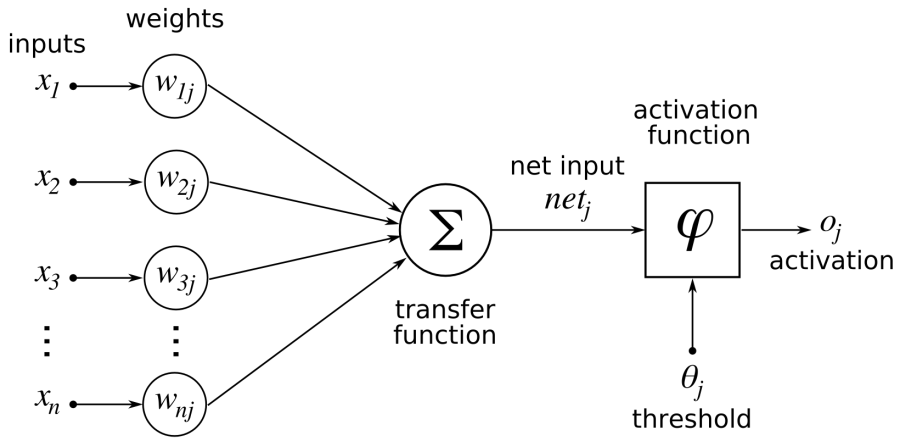

Fig. 1. *Graphical representation of one artificial neuron. Each input $x_1, ..., x_n$ is multiplied with its corresponding weight $w_{1j}, ..., w_{nj}$. The sum over all products is then passed through an activation function φ to calculate the final output o_j. Adapted from [6].*

There are multiple possible activation functions. Common choices are the binary step function, the sigmoid function or the Rectified Linear Unit (ReLU) function. Further information about activation functions can be found in [19].

2.2 Artificial Neural Networks

We can combine multiple artificial neurons to create a neural network [8]. A prevalent and easy neural network approach is the Multi-Layer Perceptron (MLP). In an MLP, neurons are grouped into multiple layers. The input layer, possibly multiple hidden layers, and one output layer. Each neuron in one layer is connected to every neuron in the following layer, therefore the output of a neuron acts as the input to every neuron of the consecutive layer. That way, the MLP forms a directed acyclic graph (see Figure 2) and falls into the category of feed-forward neural networks. With an MLP, and neural networks in general, it is possible to model and learn any linear or nonlinear function, therefore making it attractive for many application domains [1].

2.3 Training a Neural Network with Back-propagation

Neural Networks have the ability to learn through training [8]. A popular learning method is Supervised Learning (SL). In SL, many training data samples, consisting of an input and an expected output, are used to train a neural network. To solve SL tasks, we can use a method called backpropagation [21]. Backpropagation consists of multiple phases: During the so-called forward

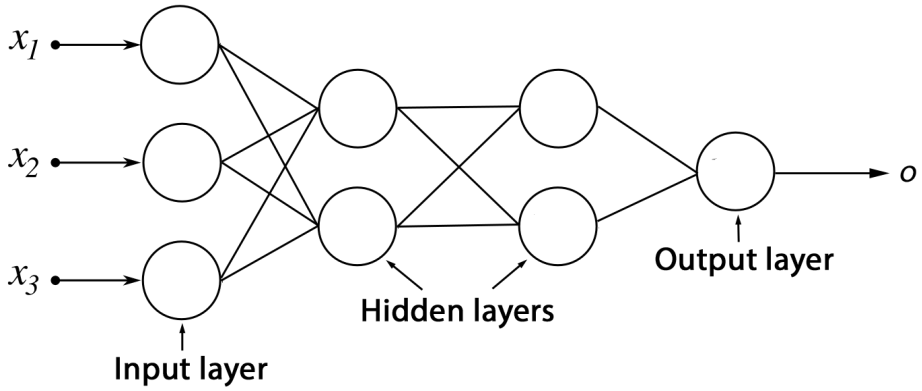

Fig. 2. *Graphical representation of a Multi-Layer Perceptron with the input x_1, x_2, x_3 and output o. The network has one input layer, two hidden layers and one output layer.*

pass, the inputs x_i from these data samples are passed through the network and the network's actual outputs o_j are calculated. Each calculated output is then compared to the expected output y_j of its data sample. A loss is computed through the difference between the desired and actual output. We most commonly use the mean squared error function to calculate the loss, which is defined as:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_j - o_j)^2.$$

Training a neural net involves minimizing the loss function by adjusting the network's parameters. This is commonly done in the so-called backward pass, using a procedure known as gradient descent. In every iteration, the algorithm calculates the gradient of the loss function L with respect to each parameter. Each parameter is then iteratively updated in the scaled negative gradient direction. The scaling factor μ determines the step size by which a parameter is updated. μ is commonly called the learning rate. Choosing the right learning rate is important when training a neural network. We will not go into any more detail about how to choose μ, however, [12] provides further information. An exemplary iteration of the gradient descent algorithm for each weight is given as:

$$w_{ij,t+1} = w_{ij,t} - \mu\frac{\delta L}{\delta w_{ij,t}},$$

where $w_{ij,t+1}$ depicts the new weight at iteration $t+1$ and $w_{ij,t}$ is the old weight at iteration t. The parameters will be updated until the maximum amount of iterations is reached, or the loss function converges to a minimum.

2.4 Summary (Prerequisite Knowledge)

Neural networks have a lot of potential in many application domains. They consist of neurons, that are combined in multiple layers. A network has an input and an output. They can learn by minimizing the difference between the expected output and calculated output through the adjustment of their parameters.

3 Introduction to Quantization

In this section, we will explore basic concepts of quantization. We will 1) take a look at how to quantize and dequantize values, 2) discuss the difference between uniform and non-uniform quantization and 3) distinguish between symmetric and asymmetric quantization.

3.1 Quantizing Values

To quantize values such as weights and activations used in neural networks, we need to define a function that maps a floating point value $x \in (\alpha, \beta)$ to its lower precision integer representation. A popular quantization function, described in [20] is the following:

$$Q(x) = Int(\frac{x}{S}) - Z, \tag{1}$$

where Q is the quantization operator, S is a floating point scaling factor and Z is the integer value that represents 0 in the quantization scheme. The $Int(\cdot)$ function maps the floating point value x to some integer through rounding. An example of the rounding function is the round-to-nearest, but other functions are also applicable, which can be seen in [16]. The scaling factor S is the same for all real values x, therefore Equation 1 is an example of uniform quantization. This means the distance between quantized values is equally spaced and thereby uniform (see Figure 3, left). It is also possible to vary the distance between each quantized value (see Figure 3, right). This is called non-uniform quantization, however, in this paper, we will only explore uniform quantization.

3.2 Symmetric and Asymmetric Quantization

The scaling factor S has been introduced in Equation 1 and is an important parameter when choosing the right quantization scheme. In [20], the scaling factor is defined as:

$$S = \frac{\beta - \alpha}{2^b - 1},$$

where b is the quantization bit width, α and β define the so-called clipping range of the real values, with α being the lower bound and β being the upper bound of the clipping range. A term used for determining the clipping range in [20] is calibration. [22] states that in practice, values x can fall outside the clipping range (α, β). In this case, the resulting quantized value would fall outside

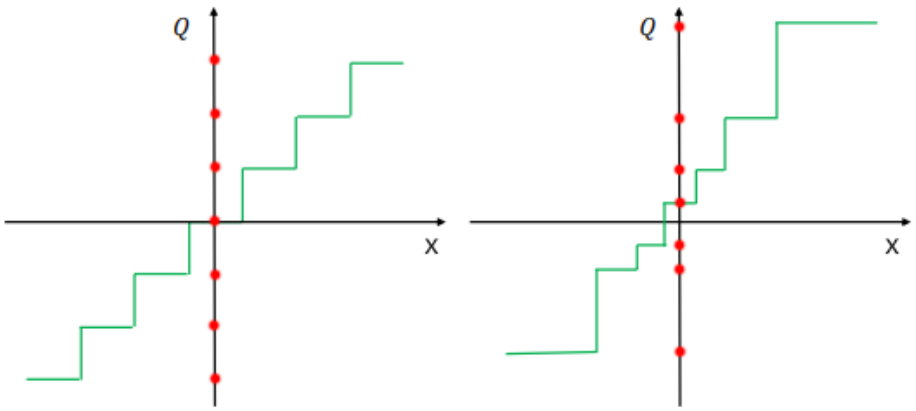

Fig. 3. *Comparison between uniform quantization (left) and non-uniform quantization (right). Real values in the continuous domain x are mapped into discrete, lower precision values in the quantized domain Q, which are marked with red bullets. Note that the distances between the quantized values are the same in uniform quantization, whereas they can vary in non-uniform quantization. Adapted from [9].*

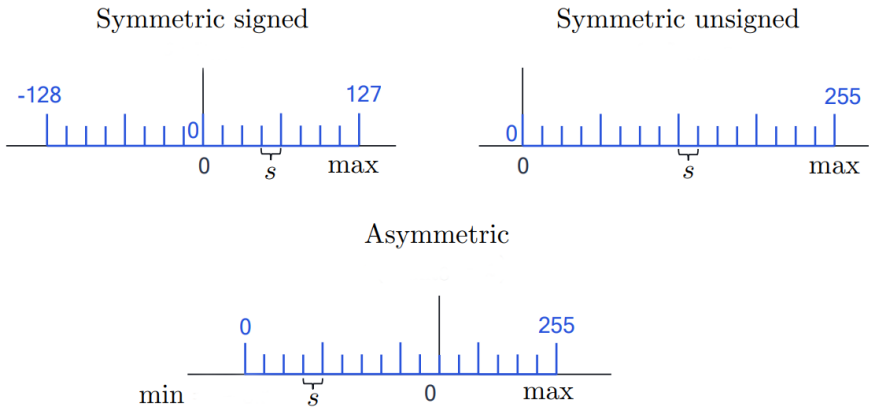

Fig. 4. *Illustration of two symmetric quantization grids (signed/unsigned) and an asymmetric quantization grid. s is the scaling factor. The floating point grid is depicted in black and the integer quantized grid is in blue. Adapted from [17].*

the integer grid range (α_q, β_q). Therefore, it is necessary to clip the quantized values. [17] proposes that, if we use signed integers, the range is defined as $(\alpha_q, \beta_q) = (-2^{b-1}, 2^{b-1} - 1)$. With unsigned integers, the range is defined as $(\alpha_q, \beta_q) = (0, 2^b - 1)$. To include the clipping function in our quantization scheme, we modify Equation 1 into:

$$Q(x) = clip\Bigg(Int(\frac{x}{S}) - Z; \alpha_q, \beta_q\Bigg), \tag{2}$$

where $clip(\cdot)$ is defined as:

$$clip(x; a, c) = \begin{cases} a, & x < a, \\ x, & a \leq x \leq c, \\ c, & x > c. \end{cases}$$

According to [9], symmetric quantization implies $\alpha = -\beta$, which can be achieved by choosing α and β with respect to the minimum and maximum values of the input: $-\alpha = \beta = \max(|x_{max}|, |x_{min}|)$ (see Figure 4, top). By using symmetric quantization, the zero-point Z is mapped to 0, which is computationally less expensive at inference time, according to [20]. Using the mapping $Z = 0$ simplifies the quantization function in Equation 2:

$$Q(x) = clip\Bigg(Int(\frac{x}{S}); \alpha_q, \beta_q\Bigg).$$

In neural networks, where the target weights or activations are imbalanced, e.g., the activation after ReLU that always has non-negative values, asymmetric quantization provides a more accurate approach. Using asymmetric quantization, the clipping range is not symmetric with respect to the origin (see Figure 4, bottom). By choosing the minimum and maximum of the input values as the clipping range, i.e., $\alpha = x_{min}$, and $\beta = x_{max}$, this asymmetric quantization can be achieved, given $-\alpha \neq \beta$ [9].

3.3 Dequantizing Values

By defining a quantization function, it is also necessary to define its inverse function, which calculates the original real value from the quantized value. [20] defines this dequantization function as:

$$\hat{x} = S \cdot (Q(x) + Z). \tag{3}$$

Using symmetric quantization, Equation 1 can again be simplified to:

$$\hat{x} = S \cdot Q(x).$$

As said in [17], it is essential to note that the dequantized value $\hat{x}$ might not equal the original real value x. This is due to the rounding function $Int(\cdot)$

introducing a bias and thereby some error that cannot be recovered by the dequantization function.

We can combine Equations 2 and 3 to get a general definition for the quantization function as:

$$\hat{x} = S \cdot \Big[clip\Big(Int(\frac{x}{S}) - Z; \alpha_q, \beta_q \Big) + Z \Big]. \tag{4}$$

3.4 Summary (Introduction to Quantization)

We introduced a quantization scheme, that can quantize real values into a smaller bit representation. We can quantize values uniformly or non-uniformly. This refers to the distance between the quantized values. We can also choose between symmetric or asymmetric quantization. This sets the clipping range of the quantized values, with symmetric quantization mapping the zero-point of the real values to 0. We can dequantize values to go from a low bit width representation to the original value, however, this introduces some bias, due to a rounding operation in the quantization process.

4 Quantization Methods

In this section, we will explore two main methods used for neural network quantization. We will 1) give an overview of Quantization-Aware Training (QAT) and 2) discuss another approach called Post-Training Quantization (PTQ). For both methods, [20] suggests using a pre-trained model, to be more effective at minimizing accuracy loss when quantizing a neural network. The key difference between QAT and PTQ is, that QAT requires retraining the network, to learn to minimize the quantization error introduced during the quantization process. PTQ can be used without retraining the model, which is beneficial when there is not enough training data available.

4.1 Quantization-Aware Training

QAT is a method for neural network quantization that involves retraining a pre-trained neural network with respect to quantized parameters such as weights and activations. QAT simulates the quantized parameters during the forward and backward pass, by injecting quantization nodes into the computation graph, to introduce the quantization error and learn to minimize it. Figure 5(b) shows the computation graph with quantization nodes. These nodes are removed during inference (see Figure 5(a)). During the backward pass, we use floating point values to calculate the gradients but run into a problem due to the quantization function, defined in Equation 4, being non-differentiable. The quantization function uses a rounding function $Int(\cdot)$ which has a gradient that is zero almost everywhere. This necessitates an approximation of the gradient. [4] proposes an

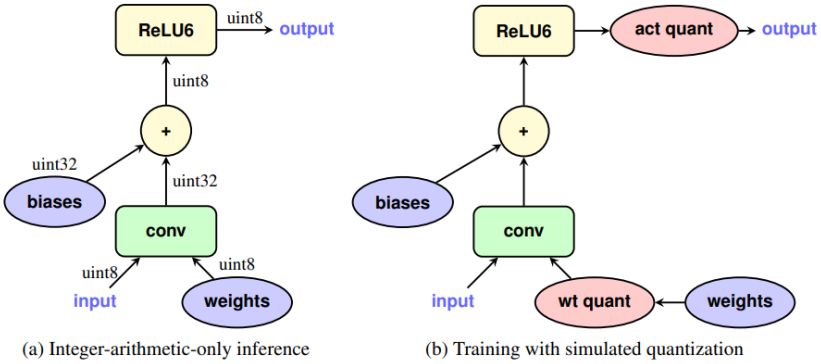

Fig. 5. *(a) shows a computation graph for integer-arithmetic-only inference of a quantized neural network. The network has been trained using simulated quantization, shown in (b). During training, quantization nodes are injected into the computation graph following the weight and activation nodes. These quantization nodes get removed during inference. Adapted from [11].*

estimator called the Straight-Through-Estimator (STE), which approximates the gradient of the rounding function as:

$$\frac{\partial}{\partial y} Int(y) = 1, \ \forall y \in \mathbb{R}. \tag{5}$$

According to [17], Equation 5 enables calculating the gradient of the quantization function 4. Note that we are using symmetric quantization, so $Z = 0$, x is the input and (α, β), (α_q, β_q) determine the clipping ranges for the float and integer values. We can now calculate the gradient of the quantization function 4 with respect to the input x as:

$$\begin{aligned}
\frac{\partial \hat{x}}{\partial x} &= \frac{\partial}{\partial x}\Big(S \cdot clip\big(Int(\frac{x}{S}); \alpha_q, \beta_q\big)\Big) \\
&= S \cdot \frac{\partial}{\partial x}\Big(clip\big(Int(\frac{x}{S}); \alpha_q, \beta_q\big)\Big) \\
&= \begin{cases} S \cdot \frac{\partial Int(x/S)}{\partial (x/S)} \cdot \frac{\partial (x/S)}{\partial x} & if \ \alpha \leq x \leq \beta, \\ S \cdot \frac{\partial}{\partial x} Int(\alpha_q / S) & if \ x < \alpha, \\ S \cdot \frac{\partial}{\partial x} Int(\beta_q / S) & if \ x > \beta, \end{cases} \\
&= \begin{cases} 1 & if \ \alpha \leq x \leq \beta, \\ 0 & otherwise. \end{cases}
\end{aligned}$$

The STE enables us to use the quantization function during the forward pass and then, as the name suggests, we can estimate straight through the quantization function and are therefore able to calculate the gradient during the backward pass (see Figure 6). [17] proposes, that we can use the STE to also calculate the gradient with respect to the other quantization parameters, Z and S. By making these quantization parameters learnable, we can achieve better accuracy [5], [7]. We calculate the gradient with respect to the scaling factor S as:

$$\begin{aligned}\frac{\partial \hat{x}}{\partial S} &= \frac{\partial}{\partial S}\Big(S \cdot clip\big(Int(\frac{x}{S}); \alpha_q, \beta_q\big)\Big) \\ &= \begin{cases} -\frac{x}{S} + Int(\frac{x}{S}) & if\ \alpha \leq x \leq \beta, \\ \alpha_q & if\ x < \alpha, \\ \beta_q & if\ x > \beta. \end{cases}\end{aligned}$$

In Section 3.1, we introduced the zero-point value Z as an integer. However, to make this parameter learnable, we first have to convert it into a real number and then apply the integer rounding operator. That way, we again get a modified quantization function as:

$$\hat{x} = S \cdot \Big[clip\Big(Int(\frac{x}{S}) - Int(Z); \alpha_q, \beta_q\Big) + Int(Z)\Big].$$

The gradient with respect to Z can now be calculated using the STE:

$$\begin{aligned}\frac{\partial \hat{x}}{\partial Z} &= \frac{\partial}{\partial Z}\Big(S \cdot \Big[clip\Big(Int(\frac{x}{S}) - Int(Z); \alpha_q, \beta_q\Big) + Int(Z)\Big]\Big) \\ &= \begin{cases} S \cdot \Big[\frac{\partial Int(x/S)}{\partial Z} - \frac{\partial Z}{\partial Z} + \frac{\partial Z}{\partial Z}\Big] & if\ \alpha \leq x \leq \beta, \\ S \cdot \Big[\frac{\partial \alpha_q}{\partial Z} + \frac{\partial Z}{\partial Z}\Big] & if\ x < \alpha, \\ S \cdot \Big[\frac{\partial \beta_q}{\partial Z} + \frac{\partial Z}{\partial Z}\Big] & if\ x > \beta, \end{cases} \\ &= \begin{cases} 0 & if\ \alpha \leq x \leq \beta, \\ S & otherwise. \end{cases}\end{aligned}$$

The above equations all use the STE, it is however possible to use different estimators. [15] proposes an alternative optimization technique, termed alpha-blending, which quantizes neural networks to low precision using

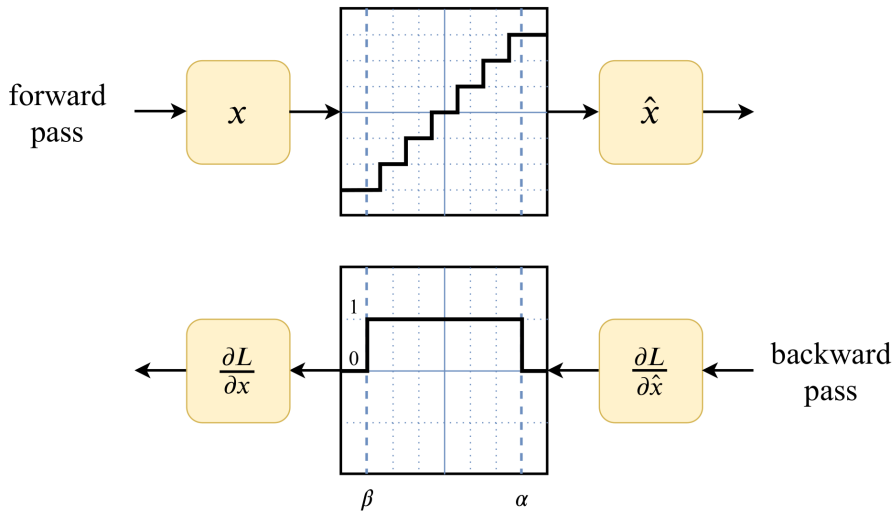

Fig. 6. *Forward and backward pass using the Straight-Through-Estimator. During the forward pass, the input x is quantized, which involves applying the non-differentiable quantization function. During the backward pass, the Straight-Through-Estimator is used to approximate the gradient of the quantization function as the identity function in the clipping range (α, β) and 0 everywhere else. Adapted from [22].*

stochastic gradient descent. Alpha-blending avoids STE approximation by replacing the quantized weight w_q in the loss function with the affine combination $(1 - \alpha)w + \alpha w_q$ of the quantized weight and the corresponding full-precision weight w with non-trainable scalar coefficient α and $(1 - \alpha)$. During training, α is gradually increased from 0 to 1. Weights are updated through the affine combination's full precision term, $(1 - \alpha)w$. That way, the model is converted from full precision to low precision progressively.

QAT offers high accuracy, however, according to [20] it comes with a trade-off. Due to the necessity to retrain the model throughout many epochs, we get high training times with correspondingly high computational retraining costs. A sufficient amount of training data is also essential to prevent the net from overfitting. In environments, where such a quantized network is deployed for long periods of time, it has been proven, that the hardware and energy efficiency gains make up for the trade-offs, that come with using QAT. [24] conducted a case study to examine resource cost and processing speed when using quantized parameters (INT8) instead of parameters in the FP32 format during training (see Table 1). Results reveal that using quantized parameters can effectively alleviate system overhead while not downgrading the model quality.

Table 1. *System performance using quantized INT8 parameters and FP32 parameters. Adapted from [24].*

	Forward Pass (ms)	Backward Pass (ms)	Per-iteration Time (ms)	Parameter Memory (MB)	Model Accuracy
FP32	95.85	140.03	240.06	18.51	97.6%
INT8	54.57	67.66	126.41	9.42	95.2%
Comparison	1.86×	2.07×	1.89×	1.96×	−2.39%

4.2 Post-Training Quantization

A different method used for quantizing neural networks is PTQ [20]. PTQ is often used when there is not enough training data available or the training data is unlabeled [9]. Unlike QAT, PTQ quantizes the network without the need to retrain it. This makes PTQ a swift method without any significant overhead. According to [20], PTQ requires to first calibrate the quantization parameters based on any available calibration data and then to quantize the model using the quantization scheme explained in Section 3. However, networks quantized using PTQ have a lower accuracy and precision than networks quantized with QAT (see Table 2). To mitigate this accuracy loss, many approaches have been explored. For example, [2] proposes a bias correction, due to an inherent bias in the mean and variance of the weight values following their quantization. Analytical Clipping for Integer Quantization is a method that analytically approximates the optimal clipping range [3]. More approaches can be found in [9].

Table 2. *A qualitative comparison of the two main quantization procedures. Adapted from [20].*

Quantization Procedure	Accuracy Loss	Quantization Time	Minimum Achievable Precision
Quantization-Aware Training	Negligible	High	≥ 1 bit [32]
Post-Training Quantization	Moderate	Low	≥ 4 bits [30]

4.3 Summary (Quantization Methods)

We explored two methods of quantization: QAT starts with a pre-trained model, which then gets retrained by injecting quantization nodes into the training process. This simulates quantized parameters, thereby enabling the network to learn the quantization error and minimize it. QAT commonly uses the Straight-Through Estimator, which is needed to perform back-propagation on

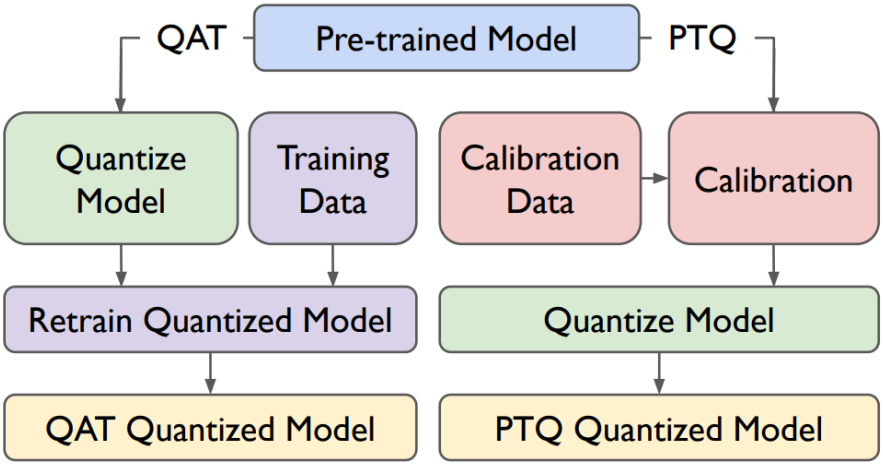

Fig. 7. *Different pipelines using QAT or PTQ for neural network quantization. The calibration data can be either a subset of the training data or a small set of unlabeled input data. Adapted from [9].*

the quantization scheme by estimating the gradient of the non-differentiable function. PTQ involves calibrating and quantizing a network without retraining it. That way, we save a lot of time, however, with a trade-off of lower precision due to the quantization error.

5 General Summary and Conclusion

In conclusion, neural network quantization, along with techniques such as QAT and PTQ, has shown to be promising for optimizing and deploying deep learning models in resource-constrained environments. Through the process of quantization, which involves reducing the precision of weights and activations, neural networks can achieve significant reductions in memory, computational requirements, and energy consumption, while maintaining acceptable levels of accuracy.

QAT has proven to be an effective approach for training neural networks to mitigate the loss of precision during quantization. By simulating the quantization process during training, the network learns to adapt to lower precision, ensuring that the quantization error is minimized. That way, we can achieve a balance between accuracy and resource efficiency.

PTQ, on the other hand, involves quantizing a pre-trained neural network without retraining. This approach provides a convenient way to apply quantization to existing models, avoiding the need for extensive retraining.

As research and development in this field continues, we can expect quantization

to play a significant role in bringing the possibilities of neural networks to devices with limited resources.

References

1. Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A., Arshad, H.: State-of-the-art in artificial neural network applications: A survey. Heliyon **4**(11), e00938 (2018). https://doi.org/https://doi.org/10.1016/j.heliyon.2018.e00938, https://www.sciencedirect.com/science/article/pii/S2405844018332067
2. Banner, R., Nahshan, Y., Hoffer, E., Soudry, D.: ACIQ: analytical clipping for integer quantization of neural networks. CoRR **abs/1810.05723** (2018), http://arxiv.org/abs/1810.05723
3. Banner, R., Nahshan, Y., Hoffer, E., Soudry, D.: ACIQ: Analytical clipping for integer quantization of neural networks (2019), https://openreview.net/forum?id=B1x33sC9KQ
4. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation (2013)
5. Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., Kwak, N.: LSQ+: improving low-bit quantization through learnable offsets and better initialization. CoRR **abs/2004.09576** (2020), https://arxiv.org/abs/2004.09576
6. Camuñas-Mesa, L., Linares-Barranco, B., Serrano-Gotarredona, T.: Neuromorphic spiking neural networks and their memristor-cmos hardware implementations. Materials **12**, 2745 (08 2019). https://doi.org/10.3390/ma12172745
7. Esser, S.K., McKinstry, J.L., Bablani, D., Appuswamy, R., Modha, D.S.: Learned step size quantization. CoRR **abs/1902.08153** (2019), http://arxiv.org/abs/1902.08153
8. Gardner, M., Dorling, S.: Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric Environment **32**(14), 2627–2636 (1998). https://doi.org/https://doi.org/10.1016/S1352-2310(97)00447-0, https://www.sciencedirect.com/science/article/pii/S1352231097004470
9. Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference (2021)
10. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine **29**(6), 82–97 (2012). https://doi.org/10.1109/MSP.2012.2205597
11. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
12. Jacobs, R.A.: Increased rates of convergence through learning rate adaptation. Neural Networks **1**(4), 295–307 (1988). https://doi.org/https://doi.org/10.1016/0893-6080(88)90003-2, https://www.sciencedirect.com/science/article/pii/0893608088900032
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger,

K. (eds.) Advances in Neural Information Processing Systems. vol. 25. Curran Associates, Inc. (2012), https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
14. Liu, Z., Cheng, K., Huang, D., Xing, E.P., Shen, Z.: Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. CoRR **abs/2111.14826** (2021), https://arxiv.org/abs/2111.14826
15. Liu, Z.G., Mattina, M.: Learning low-precision neural networks without straight-through estimator(ste) (2019)
16. Nagel, M., Amjad, R.A., van Baalen, M., Louizos, C., Blankevoort, T.: Up or down? adaptive rounding for post-training quantization (2020)
17. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T.: A white paper on neural network quantization (2021)
18. Schmidhuber, J.: Deep learning in neural networks: An overview. CoRR **abs/1404.7828** (2014), http://arxiv.org/abs/1404.7828
19. Siddharth Sharma, Simone Sharma, A.A.: Activation functions in neural networks. International Journal of Engineering Applied Sciences and Technology **4**(12), 310–316 (2020)
20. Weng, O.: Neural network quantization for efficient inference: A survey (2023)
21. Werbos, P.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE **78**(10), 1550–1560 (1990). https://doi.org/10.1109/5.58337
22. Wu, H., Judd, P., Zhang, X., Isaev, M., Micikevicius, P.: Integer quantization for deep learning inference: Principles and empirical evaluation (2020)
23. Zhang, Z.: Artificial Neural Network, pp. 1–35. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-67340-0$_1$, https://doi.org/10.1007/978-3-319-67340-0_1
24. Zhou, Q., Guo, S., Qu, Z., Guo, J., Xu, Z., Zhang, J., Guo, T., Luo, B., Zhou, J.: Octo: INT8 training with loss-aware compensation and backward quantization for tiny on-device learning. In: 2021 USENIX Annual Technical Conference (USENIX ATC 21). pp. 177–191. USENIX Association (Jul 2021), https://www.usenix.org/conference/atc21/presentation/zhou-qihua