Quantifying gradients in deep Convolutional Neural Networks with the DoReFa-Net

Felix Ferber
Supervisor: Haibin Zhao

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

Abstract. In this paper we look into the work of DoReFa-Net. A method introduced to quantize gradient parameters of deep Convolutional Neural Networks with parameter of arbitrary bit-width. We ease the understanding for the proposed methods and ideas by providing the needed basic knowledge in this field in a brief manner, and explain the important related research which laid grounds for the DoReFa-Net. This paper is targeted for a broad audience with basic knowledge in computer science and aims to make the work of Zhou Shuchang and their team more accessible for students and those interested.

Keywords: Neural Network · Quantization · Review

1 Introduction

To introduce you to the general topic of DoReFa-Net [12] we need to provide some basic foundations. We dedicate this section to explain, in a brief manner, some needed basic knowledge. With the goal in mind, to not get distracted by confusing details later on.

We will start off by detailing some definitions and explanations regarding Neural Networks and explaining in that context the structure and generally neurons. We will dive into Convolutional Neural Network and deep Convolutional Neural Network, to understand the kind of Neural Networks DoReFa-Net is applicable to.

Following those explanations we will briefly explain the general training process and explain the terms forward- and backward-pass in this context. And in that manner we will also explain data sets and the important data sets mentioned in the works.

In the terms of Neural Network quantification we don't get around the terms size and computational complexity which we will explain in the last part of this section.

In the next Section we will use all those basics to introduce briefly some related and important works close to DoReFa-Net before we then detail the works of Zhou Shuchang and their team [12].

1.1 Neural networks

A Neural Network is a computational model inspired by the structure and function of the human brain. It consists of interconnected nodes, called neurons, organized in layers. Neural Networks are used for various tasks, such as pattern recognition, classification, regression, and optimization.

Neurons, synapses and gradients

In the context of Neural Networks, a neuron is a fundamental unit, that takes a group of weighted inputs, applies an activation function, and returns an output. Neurons are connected to each other through synapses, which can be thought of as roads in a Neural Network. Each connection between two neurons has a unique synapse with a unique weight and activation attached to it. Gradients are a fundamental concept in the training of Neural Networks. They represent the rate of change of a function with respect to the parameter weights of the network. Gradients are used in algorithms to update the parameters of the network.

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a specialized type of Neural Network designed for processing structured arrays of data, such as images. CNNs are widely used in computer vision tasks and have become the state-of-the-art for many visual applications, such as image classification and object detection.

Deep Convolutional Neural Networks

A Deep Convolutional Neural Network (DCNN) refers to a CNN with multiple convolutional layers stacked on top of each other. DCNNs have been shown to achieve better performance on complex visual tasks compared to shallow Convolutional Neural Networks with fewer layers.

1.2 Training

Training in the context of Neural Networks refers to the process of optimizing the model's parameters such that it can make accurate predictions on new, unseen data. During the training process, the model is presented with a set of labeled training examples and learns to adjust its parameters based on the errors it makes. The goal is to minimize the difference between the predicted outputs and the true outputs.

During training, the model goes through multiple iterations, also known as epochs, where it updates its parameters using an algorithm. The training process involves two main steps: the forward pass and the backward pass.

Forward pass

The forward pass, also known as forward propagation, is the calculation and storage of intermediate variables and outputs for a Neural Network. It involves processing the input data through the layers of the network, from the input layer to the output layer, to obtain the predicted outputs. Each layer performs a set of computations, typically involving matrix multiplications and activation functions, to transform the input data.

In the forward pass, the input data is multiplied by the weights of the connections between the neurons in each layer, and the resulting values are passed through activation functions to introduce non-linearity. This process is repeated for each layer until the final output is obtained. The intermediate variables and outputs calculated during the forward pass are stored for later use in the backward pass.

Backward pass

The backward pass, also known as backward propagation, is the process of calculating the gradients of the model's parameters with respect to the loss function. It involves traversing the computational graph in the reverse direction, from the output layer to the input layer, to update the parameters based on the errors made during the forward pass. These errors are represented as an error function or loss function.

In the backward pass, the gradients of the loss function with respect to the outputs of each layer are calculated using calculus. These gradients are then used to update the weights and biases of the model using an optimization algorithm like gradient descent. The gradients are propagated backwards through the layers of the network, hence the name "backpropagation".

1.3 Datasets

Datasets refer to a collection of data that is organized and structured for use in machine learning and data analysis. Datasets can vary in size and complexity, ranging from small datasets used for testing and experimentation to large datasets used for training deep learning models. Datasets can contain different types of data, including numerical, categorical, and text data. Datasets are crucial for building and evaluating machine learning models, as they provide the necessary information for the models to learn patterns and make predictions.

SVHN

SVHN [9] stands for Street View House Numbers. It is a real-world image dataset that consists of images of house numbers taken from Google Street View. The dataset is widely used for training and evaluating machine learning models for digit recognition tasks. Each image in the SVHN dataset contains a number from 0 to 9.

ImageNet

ImageNet [4] is a large-scale image dataset that contains millions of labeled images. It is widely used in the computer vision community for training and evaluating deep learning models. The dataset covers a wide range of object categories, including animals, vehicles, and everyday objects. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition that benchmarks the performance of models on ImageNet.

1.4 Complexity

In the context of Neural Networks, complexity refers to the computational resources required to train and run a Neural Network model. It involves measuring both the time complexity and space complexity of the algorithms used in Neural Networks. Time complexity refers to the amount of time it takes to train and make predictions with a Neural Network, while space complexity refers to the memory storage required during the training and inference processes. The complexity of a Neural Network is influenced by factors such as the number of layers, the number of neurons in each layer, and the size of the input data. Increasing the complexity of a Neural Network can lead to more accurate predictions but may also require more computational resources.

2 Recent efforts

We looked into the basic concepts for Neural Networks helping us to understand the DoReFa-Net. In this section we will briefly explain selected research topics preceding DoReFa-Net, which were crucial for the work of Zhou Shuchang and their team [12]. Beginning with the concepts of quantification and sparsification for reducing complexity of Neural Networks, followed by models which the DoReFa-Net was derived from. After that we will start introducing the concept of DoReFa-Net.

2.1 Quantification

Quantization in the context of reducing the complexity of a Neural Network involves representing the network's parameters and activations with reduced precision, such as using fewer bits to represent numerical values. By quantizing these values, according to the team of Wu [11], the memory requirements and computational complexity of the network can be significantly reduced. This can lead to benefits like faster inference, reduced energy consumption, and improved deployment on devices with limited resources, while still maintaining acceptable accuracy levels.

2.2 Sparsification

Sparsification in the context of reducing the complexity of a Neural Network involves inducing sparsity in the network's parameters or activations, meaning that a significant portion of the values become zero. By removing or "pruning" these zero values, the network's size and computational requirements can be reduced. Sparsification techniques, like in the work of Han and their team [6], aim to retain the most important connections or activations while discarding the less significant ones, allowing for more efficient memory usage, faster computations, and potential acceleration through sparse matrix operations.

2.3 AlexNet

AlexNet [8] is a CNN architecture that was designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. It was the winning entry in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. AlexNet is considered a breakthrough in the field of computer vision and deep learning, as it demonstrated the effectiveness of DCNNs for image classification tasks.

2.4 Binary Neural Network

Binarized Neural Networks (BNNs), by the team of Courbariaux [3], are a type of Neural Network model that use binary values (usually -1 and +1) for both weights and activations, instead of traditional real-valued representations. This binary representation significantly reduces memory usage and accelerates computations by replacing costly multiplications with simple bitwise operations like XNOR and popcount.

In BNNs, the binarization process typically involves a step called "binarization function," which converts the real-valued weights and activations into binary values. This function can be deterministic, such as sign function, or probabilistic, where the probabilities of -1 and +1 are learned during training. During forward propagation, the binarized weights and activations are used for computations, and the results are accumulated using bitwise operations.

2.5 XNOR-Net

XNOR-Net [10] is a Neural Network model that extends the idea of BNNs by leveraging the XNOR operation. In XNOR-Net, both the weights and binary activations are binarized to -1 and +1, similar to BNNs, but the convolutional operations are replaced with XNOR and bit-count operations.

In XNOR-Net, the key innovation lies in performing convolutional operations using XNOR and bit-count operations. The multiplication operation between binary weights and binary inputs is effectively replaced with the simple bitwise XNOR operation, making the computations significantly faster. The result is then passed through a bit-count operation to accumulate the binary convolutional outputs.

3 DoReFa-Net

In this section we will discuss the DoReFa-Net, starting with the challenges, which were in mind of Zhou Shuchang and their team [12]. We elaborate on the proposed ideas in more detail leveraging the previous explanations. Experiments done in the scope of DoReFa-Net will be discussed briefly and we highly encourage you to look into the original work for more details in that regard.

3.1 Introducing DoReFa-Net

DoReFa-Net is a method for training CNNs with low bitwidth weights and activations using low bitwidth parameter gradients. This allows for significant reductions in the memory and computational requirements of CNNs, while still achieving comparable accuracy to 32-bit counterparts.

DoReFa-Net works by stochastically quantizing the parameter gradients to low bitwidth numbers before they are propagated to the convolutional layers. This is done during the backward pass of the training algorithm. As a result, the convolutions during the forward and backward passes can now operate on low bitwidth weights and activations/gradients respectively. This allows DoReFa-Net to use bit convolution kernels, which are much more efficient than traditional convolution kernels.

The main advantages of DoReFa-Net are its efficiency and accuracy. DoReFa-Net can significantly reduce the memory and computational requirements of CNNs, while still achieving comparable accuracy. This makes DoReFa-Net a promising approach for deploying CNNs on resource-constrained devices, such as mobile phones and embedded systems.

3.2 The contributions of DoReFa-Net

In this chapter we will list the contributions made by the paper of Zhou Shuchang and their team [12], without going into too many details. Subsequently we will start detailing the method of DoReFa-Net which includes detailed explanations to ease the understanding and provide the context for this work.
The paper on DoReFa-Net [12] detailed their contribution as follows:

- A generalized method of BNNs [3] such that it allows creating a DoReFa-Net, which is a CNN with customizable bitwidth for activation, weight and gradient respectively. This allows operating with low bitwidth kernels for accelerated backward and forward passes during training processes.
- Opening an accelerated way of training low bitwidth Neural Networks on Hardware like CPU, FPGA, ASIC and GPU. And even reducing costs by reducing considerably the energy consumption during training especially on FPGA and ASIC.

– Exploration and experiments on different bitwidth configurations of the weights, activation and gradient parameters for DoReFa-Net. Given the 1-bit weight, 1-bit activation and 2-bit gradient configuration which achieved 93% accuracy on the SVHN dataset.
– A release of a pre-trained DoReFa-Net model in TensorFlow [1], derived from AlexNet [8] trained on the SVHN [9] dataset.

Side note:
Field-Programmable Gate Arrays (FPGAs) are a programmable integrated circuit that can be configured after manufacturing, offering flexibility and adaptability for various applications. Application-Specific Integrated Circuits (ASICs) are a customized integrated circuit designed and optimized for a specific task or application, providing high performance and power efficiency. FPGAs can be reprogrammed and are therefore suitable for prototyping, rapid development, and applications requiring flexibility. ASICs are tailored for specific functions during manufacturing and cannot be reconfigured, offering optimized performance and power efficiency.

3.3 The formulation of DoReFa-Net

Now that we have introduced the idea of DoReFa-Net we will go into detail of the formulation. In this we will explain the bit convolution kernel, which is the base for the low bit operations of the parameters, in regards to the DoReFa-Net. We will detail how this is leveraged and explain the Straight Though Estimator and its use, before we go into detail of the quantization of each weight, activation and gradient parameter.

Following this chapter we will detail the pseudo code algorithm on how a DCNN could be trained using the DoReFa-Net method.

Bit convolution Kernels

Bit convolution kernels are convolutional filters that operate at the bit level. These small matrices help the Neural Network understand spatial patterns in the input. In these filters the convolution operation is performed on binary representations of the input data rather than the traditional numerical values.

In equation 1 we specify the 1-bit dot kernel. This can also be used to calculate the dot product in general and consequently convolution, for low bitwidth fixed-point integers.

$$\mathtt{x} \cdot \mathtt{y} = bitcount(and(\mathtt{x}, \mathtt{y})), x_i, y_i \in \{0, 1\} \forall i. \tag{1}$$

Which we are interested in because fixed-point number arithmetic reduces memory requirements and computational complexity compared to floating-point arithmetic. We can use this for more efficient hardware implementation, resulting in faster and more power-efficient inference. Additionally can we exploit fixed-point arithmetic for quantization of network parameters.

To show this Zhou Shuchang and their team [12] created the following example. Assume x is a sequence of M-bit fixed-point integers s.t. $x = \sum_{m=0}^{M-1} c_m(x)2^m$ and y a series of K-bit fixed-point Integers s.t. $y = \sum_{k=0}^{K-1} c_k(y)2^k$, where $c_m(x)_{m=0}^{M-1}$ and $c_k(y)_{k=0}^{K-1}$ are vectors containing only the number 0 and 1.

The dot product of x and y can be computed by the bit-wise operation as:

$$x \cdot y = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} bitcount[and(c_m(x), c_k(y))] \tag{2}$$

$$c_m(x)_i, c_k(y)_i \in \{0, 1\} \forall i, m, k. \tag{3}$$

The above equations have computational complexity on $O(NK)$ and are therefore directly proportional to the bitwidth of x and y

These equations extend the introduced 1-bit dot product kernel of equation 1 to a bit convolution Kernel for bit vectors of arbitrary length.

Straight Through Estimator

In order to represent real numbers with a set of finite numbers, we can naively achieve this with a step function, which maps numbers in specific ranges to a set value. Problems arise with this in the back propagation, simply because almost all sections of a stepping function have a gradient of zero.

To circumvent this problem DoReFa adopts the Straight-Through-Estimator method [7] [2]. This method allows DoReFa-Net to define arbitrary forward and backward operations.

The Straight Through Estimator extensively used is $quantize_k$, which is defined to quantize the real number input $r_i \in [0, 1]$ on an k-bit number output $r_o \in [0, 1]$.

The definition of the Straight-Through-Estimator is as follows:
Forward operation:

$$r_o = \frac{1}{2^k - 1} round((2^k - 1)r_i) \tag{4}$$

Backward operation:

$$\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}. \tag{5}$$

The r_o in the forward operation is just the closest fraction to r_i in the form of $\frac{t}{2^k-1}$ for $t \in \mathbb{N}$ and $t < 2^k$. For the backward operation c denotes the function 4 from the forward pass, which makes $\frac{\partial r_i}{\partial r_o}$ not a differentiable function. But we leverage the fact the r_i is approximately equal to r_o and use the well defined gradient $\frac{\partial c}{\partial r_o}$ as an approximation for $\frac{\partial c}{\partial r_i}$.

Using this construction, we can represent a real number with k-bits. Also, with r_o being a fixed point integer, we can use the defined dot product function 2 to calculate the dot product of two such k-bit real numbers, by properly tweaking the equation.

Low bitwidth quantization

In this section we will look into the quantization of weights, activation and gradient parameters respectively. This will introduce the proposed method of gradient quantization. Each chapter and equation has explanations to ease the understanding. Each of the quantized parameters will use a Straight Through Estimator introduced in the previous chapter.

The quantization of the weight parameters

Previous work, like the BNN [3] or XNOR-Net [10] already introduced the binarization of the weight parameters using the Straight Through Estimator. In the BNN the Straight Through Estimator to binarize the weights looked like this:
Forward operation:

$$r_o = sign(r_i) \tag{6}$$

Backward operation:

$$\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o} \mathbb{I}_{|r_i| \leq 1}. \tag{7}$$

Here the $\mathbb{I}_{condition}$ stands for the index function, which is defined to equal to 0 if the condition in the subscript is $false$ and equals 1 otherwise.

The function $sign(r_i)$ from equation 6 is defined as $sign(r_i) = 2\mathbb{I}_{r_i \geq 0} - 1$, which only returns the values $\{-1, 1\}$.

The XNOR-Net has a slightly different binarization equation for their weight parameters. The difference being scaling the weights after being binarized.
The Straight Through Estimator looks as follows:
Forward operation:

$$r_o = sign(r_i) \times \mathbf{E}_F(|r_i|) \tag{8}$$

Backward operation:

$$\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}. \tag{9}$$

The function $\mathbf{E}_F$ in equation 8 is the absolute mean of all the input weights and used as the scaling factor. The reasoning for the decision to introduce a scaling factor is that it increases the value range for the weights while still exploiting bit convolution kernels.

This might seem like a preferable decision, in terms of DoReFa-Net however, it is left out. Solely for the fact that we find it impossible to exploit bit convolutions between gradient and weight in the back propagation.

We instead use constant scaling instead of channel-wise scaling. The Straight Through Estimator used for DoRoFa-Net binary weight parameters is as follows:
Forward operation:

$$r_o = sign(r_i) \times \mathbf{E}(|r_i|) \tag{10}$$

Backward operation:

$$\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}. \tag{11}$$

Weight parameters which are not binary and instead use the k-bit representation for their weights, with $k > 1$, use the following Straight Through Estimator f_ω^k instead:
Forward operation:

$$r_o = f_\omega^k = 2quantize_k(\frac{tanh(r_i)}{2max(|tanh(r_i)|)} + \frac{1}{2}) - 1 \tag{12}$$

Backward operation:

$$\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}. \tag{13}$$

Important to note is that the $\frac{r_o}{r_i}$ in 13 is well defined due to the used function in 12 being already the earlier defined Straight Through Estimator called $quantize_k$

In the function 12 the $tanh$ is used to limit the value range of the weight to $[-1, 1]$ before the quantization of the k-bit. The value of $\frac{tanh(r_i)}{2max(|tanh(r_i)|)} + \frac{1}{2}$ can only be in the range of $[0, 1]$, with the max-function taking the maximum over all weights of the current layer. The $quantize_k$ function, from equation 4, then turns this into an k-bit fixed point number in the $[0, 1]$ range. Finally we use the same transformation as the $sign$ function used in the definition from the equation 6 in 3.3, to transform the range to $[-1, 1]$.

We note as alternative way of binarizing weights by setting $k = 1$ in equation 12. Nonetheless Zhou Shuchang and their team [12] found the difference to be insignificant in their experiments.

The quantization of the activation parameters

In this section we will detail the approach to low bitwidth activations parameters which are input to the convolutions. This makes them crucially important when replacing the computational intensive floating-point operations.

In earlier research of BNNs [3] and XNOR-Net [10] the activation parameters were binarized in the same way as the weight parameters. This approach was found by Zhou Shuchang and their team [12] either non reproducible or bound to loss in severe amounts of accuracy especially when training against an ImageNet [4] models like AlexNet [8].

That's the reason why we use a Straight Through Estimator for input activations r of each weight layer. For this we assume the output of the previous layers to be bounded by their respective activation function h, which ensures us that the value of $r \in [0, 1]$. The DoReFa-Net quantization of the activations r to k-bit is then:

$$f_\alpha^k = quantize_k(r). \tag{14}$$

The quantization of the gradient parameters

Contrary to the deterministic approach of the activation and weights for the quantization, we find an stochastic approach to quantize the gradient parameter as necessary. As shown in the experiments of the Gubta and their team [5] already for 16-bit weights and gradients.

The quantization of gradients is different already in regard to the unbounded nature, therefore making the value range significantly greater than those of the activations.

Looking at Equation 14, we mapped activation simply onto $[0, 1]$, by using the earlier defined nonlinear differentiable function and passing our values through. Such a function, does not exist for gradients. For that reason we use the following function for the quantization of k-bit gradients:

$$\bar{f}_{\gamma}^{k} = 2max_0(|dr|)(quantize_k(\frac{dr}{2max_0(|dr|)} + \frac{1}{2}) - \frac{1}{2}). \tag{15}$$

In this equation it is $dr = \frac{\partial c}{\partial r}$, the back-propagated gradient of r for the corresponding layer. The maximum is taken over all axes of the gradient tensor dr, with exception for the mini-batch axis, so each mini-batch and it's axis will get its own scaling factor when applying this function.

The function above maps the gradient into the range $[0, 1]$, quantizes those values using the $quantize_k$ function and scales the function back to its original range.

This quantization can introduce strong biasing of the values and to compensate for that we introduce an additional noise function $N(k) = \frac{\sigma}{2^k - 1}$ where $\sigma \sim Uniform(-0.5, 0.5)$ which is the uniform distribution. It is noted that in this definition the endpoints of the uniform distribution are not clipped, due to the fact that those are almost never being attained.

This noise function has the same magnitude as the possible quantization error. In the experiments of Zhou Shuchang and their team [12], they found it to be critical to add this noise to achieving good performance. Adding this noise into the equation 15 we get the following:

$$f_{\gamma}^{k} = 2max_0(|dr|)(quantize_k(\frac{dr}{2max_0(|dr|)} + \frac{1}{2} + N(k)) - \frac{1}{2}). \tag{16}$$

Due to the Gradient only being quantized on the backward pass. We use the following Straight Through Estimator on the output of each convolutional layer:
Forward operation:

$$r_o = r_i \tag{17}$$

Backward operation:

$$\frac{\partial c}{\partial r_i} = f_{\gamma}^{k}(\frac{\partial c}{\partial r_o}). \tag{18}$$

3.4 The algorithm for DoReFa-Net

In this section we will see an example algorithm to train a DCNN with arbitrary bitwidth using the quantification methods introduced previously. Following that we will explain the workings of that algorithm a little more in detail, before we end this chapter with some exceptions in this method and a note on how to reduce run-time memory usage by fusing nonlinear functions with rounding.

Starting off with the algorithm taken from the original work from DoReFa-Net [12]:

Algorithm 1 Training a L-Layer DoReFa-Net with W-bit weights and A-bit activations using G-bit gradients. Weights, activations and gradients are quantized according to equation 12, 14 and 16, respectively

```
Require: a minibatch of input and targets $(a_0, a^*)$, previous Weights $W$, learning rate $\eta$
Ensure: updated weights $W^{t+1}$
    {1. Computing the parameter gradients:}
    {1.1 Forward propagation:}
 1: for $k = 1$ to $L$ do
 2:     $W_k^b \leftarrow f_\omega^W(W_k)$
 3:     $\widetilde{a}_k \leftarrow \texttt{forward}(a_{k-1}^b, W_k^b)$
 4:     $a_k \leftarrow h(\widetilde{a}_k)$
 5:     if $k < L$ then
 6:         $a_k^b \leftarrow f_\alpha^A(a_k)$
 7:     end if
 8:     Optionally apply pooling
 9: end for
    {1.2 Backward Propagation:}
    Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$.
10: for $k = L$ to $1$ do
11:     Back-propagate $g_{a_K}$ through activation function $h$
12:     $g_{a_k}^b \leftarrow f_\gamma^G(g_{a_k})$
13:     $g_{a_{k-1}} \leftarrow \texttt{backward\_input}(g_{a_k}^b, W_k^b)$
14:     $g_{W_k^b} \leftarrow \texttt{backward\_weight}(g_{a_k}^b, a_{k-1}^b)$
15:     Back-propagate gradients through pooling layer if there is one
16: end for
    {2. Accumulate the parameters gradients:}
17: for $k = 1$ to $L$ do
18:     $g_{W_k} = g_{W_k^b} \frac{\partial W_k^b}{\partial W_k}$
19:     $W_k^{t+1} \leftarrow \texttt{Update}(W_k, g_{W_k}, \eta)$
20: end for
```

To ease the understanding we will recap the algorithm in more details here. This is purely to ease understanding this simple, but dense algorithm.

To begin in the forward propagation, we assign values to the parameter weights vector of this minibatch using the quantization function from equation

12 (`line2`). We calculate the new activations using an arbitrary `forward` function(`line3`), which now can operate on low bitwidth and fixed-point integers. Followed by assigning the new activation value after passing through the activation function h (`line4`) and finally the quantization of the activations of the layer using the function from equation 14 (`line6`). Important to note is that the layer 0 and L will never be quantized here, but we will come back to this in the next chapter.

The backward propagation starts with calculating the first gradient of the layer. Using the resulting gradient we start the loop with back propagating it through the activation function h (`line11`). We quantize the result using the quantization function from equation 16 and assign it to the gradient for this minibatch(`line12`). Then we use arbitrary $\texttt{backward}_{\texttt{i}}\texttt{nput}$ and $\texttt{backward}_{\texttt{w}}\texttt{eight}$ to firstly get the gradient for the next iteration (`line13`) and get the gradient which we need to update the weights parameters(`line14`).

In the last loop we collect the gradients (`line18`) from the minibatches and apply the arbitrary `Update`() function to get our updated weights parameters.

Computational intense are the arbitrary functions `forward`, $\texttt{backward}_{\texttt{i}}\texttt{nput}$ and $\texttt{backward}_{\texttt{w}}\texttt{eight}$, which all possibly are mapped on low bitwidth numbers and fixed-point integers with affine transformations. Which means those expensive operations are now accelerateable significantly by the fixed-point integer dot product kernel 2.

The First and last layers

As the previous explanation suggested, in a DCNN, the first and last layers appear to be different from the rest. As they are the layers interfacing the input and output. For the first layer it is often an image which might contain non redundant features in 8-bit or higher than the bitwidth of the features. As for the outputs from the last layer, they produce one hot vector. Which are $1 \times N$ vectors with a singular 1 and 0's on every other position, those are used for mapping to a vocabulary from a table. By this definition they are close to bit vectors and converting those layers to low bitwidth counter parts is left open in the work of DoRoFa-Net.

Arguments for not reducing the computational complexity in those layers are mainly based on observed degradation of accuracy, while looking into the sparsification of the convolutions in a Neural Network by Han and their team [6]. This observation and the fact that for a DCNN the first and last layer alone are a small fraction of the computational complexity, the quantization is left open in the experiments done by Zhou Shuchang and their team [12].

However even though the input of the first layer and the output of the last layer are fully connected. The parameters used by the preceding and following layers are again quantized by definition of the algorithm.

Reducing Run-time memory usage

Reducing Memory during run-time is one of the motivations for creating a low bitwidth Neural Network and is, as shown earlier, possible. Nevertheless we have to point out that a naive implementation of Algorithm 1 would store activations from the activation function $h(a_k)$ in full-precision and therefore use a lot of memory during run-time. Especially when using floating-point arithmetic, which also would introduce a lot of computational complexity back into the algorithm, due to it's non-bitwise nature.

To avoid this we can fuse `line3`, `line4` and `line6` with the goal in mind not storing the intermediate floating-point results in full precision, which would inevitably lead to non-bitwise arithmetic. For that we have to point out the fact, that if the function h is monotonic, the function $f_\alpha \cdot h$ is also monotonic. Which again implies that the few values for a_k^b, which in of themselves are non-overlapping, correspond to also non-overlapping ranges of a_k. Keeping this in mind, we can implement $a_k^b = f_\alpha(h(a_k))$ with computational simple fixed-point number comparisons, resulting in the desired avoidance of intermediate floating-point results.

Similarly can we fuse `line11`, `line12` and from the preceding iteration `line13` to again avoid the intermediate result of g_{a_k}, even though more complex when there are intermediate pooling layers, we can do the fusion nevertheless. As the function $quantize_k$ commutes with the max function, such that

$$quantize_k(max(a,b)) = max(quantize_k(a), quantize_k(b)) \tag{19}$$

implies that $g_{a_k}^b$ can be generated from g_{a_k} only using fixed-point number comparisons.

3.5 Experiments

In this chapter, we provide a brief overview of the experiments conducted by Zhou Shuchang and their team [12] and summarize some notable results. We strongly recommend referring to their work for more comprehensive details and data related to these experiments.

The experiments involved multiple sets of data with varying configuration spaces for the models, where each configuration represented different bitwidths of weight, activation, and gradient parameters. The first set of configurations focused on accuracy, storage size ranking, and complexity ranking for different types of configurations. Each configuration was trained on four models with varying channel counts.

For this set of configurations, the SVHN [9] dataset was utilized. We observed that the accuracy degradation was not significant for model A, which had full connectivity in all seven layers, when the gradient's bitwidth (G) was greater than or equal to 4. However, there were exceptions to this trend, as seen in the configuration data. For instance, the (1, 1, 2) configuration achieved accuracies of approximately 93.4%, 92.4%, 91%, and 80.3% for models A, B, C, and D, respectively. Conversely, the (1, 4, 2) configuration, which would suggest higher accuracy with a higher bitwidth for one parameter, achieved accuracies of 81.5%, 89.8%, 91.1%, and 86.8% for the same models.

In general, we observed a trend where models with greater connectivity between layers tended to have higher accuracy. Additionally, increasing the gradient bitwidth led to improved accuracy, but the improvement became insignificant after 8 bits.

The second set of configurations utilized the ILSVRC12 [4] image classification dataset, which contains around 1.2 million natural images from approximately 1,000 object categories. Here, the focus was on different gradient bitwidths for 1-bit weights and activation with bitwidths of $A \leq 4$. We observed a clear improvement in accuracy with higher bitwidths and even similarity in accuracy between configurations with 32-bit and 6-bit gradients.

We recommend referring to the work of Zhou Shuchang and their team [12] for more detailed explanations and extensive data on these experiments.

4 Conclusion

In conclusion, the paper by Zhou Shuchang and their team [12] presents a highly impactful work that introduces a well-formulated and easily understandable idea. The paper effectively contextualizes the provided information and delivers a clear message. With over 2,000 citations to date, this work is widely recognized and highly influential in the field. We strongly recommend reading the original paper if you find the topic intriguing.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
2. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
3. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1 (2016)
4. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848
5. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: International conference on machine learning. pp. 1737–1746. PMLR (2015)
6. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (2016)
7. Hinton, G., Srivastava, N., Swersky, K.: Neural networks for machine learning (2012)

8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks (alexnet)
9. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)
10. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks (2016)
11. Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4820–4828 (2016)
12. Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR **abs/1606.06160** (2016), http://arxiv.org/abs/1606.06160