Quantized Back Propagation for Efficient Deep Learning

by

Kevin Kepp
366263

Master's Thesis

reviewed by

Prof. Dr. Klaus-Robert Müller
Technische Universität Berlin

and

Prof. Dr. Thomas Wiegand
Technische Universität Berlin
Fraunhofer Heinrich-Hertz-Institut

supervised by

Dr. Wojciech Samek
Fraunhofer Heinrich-Hertz-Institut

October 19, 2018

Declaration

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, October 19, 2018

_____

Kevin Kepp

Abstract

This thesis introduces a novel approach to leveraging quantization to improve the efficiency of training neural networks. Neural networks are successful but computationally expensive learning systems that are trained with stochastic optimization algorithms on large amounts of data. The noise in the stochastic training process was shown to be beneficial for converging to network configurations that generalize well, which means they provide good predictions for new data samples. Our motivation is to develop a quantization scheme that mimics the specific characteristics of this noise. We thus leverage a stochastic quantization function that exhibits controllable noise statistics while inducing sparsity on intermediate results during the training algorithm. This sparsity can be exploited by computing efficient sparse matrix multiplications, which greatly reduce the required operations and thus the computational cost of the training algorithm. We prove that for training certain types of networks, uniform quantization with an additive stochastic signal possesses the desired properties of controllable quantization noise and high induced sparsity. Experiments on popular image classification tasks confirm the theoretical properties of our method and show that it outperforms approaches with deterministic quantization schemes by achieving better generalization-efficiency ratios.

Zusammenfassung

Diese Arbeit stellt einen neuen Ansatz für effizientes Training von neuronalen Netzen durch den Einsatz von Quantisierung vor. Neuronale Netze sind erfolgreiche aber rechenintensive Lernsysteme, die mit stochastischen Optimierungsalgorithmen auf großen Datenmengen trainiert werden. Das stochastische Rauschen des Trainingsprozesses ist nachweislich hilfreich bei der Konvergenz zu Netzwerkkonfigurationen die gut generalisieren, das heißt gute Vorhersagen auf neuen Datensätzen treffen können. Diese Eigenschaft gibt Anlass für die Entwicklung einer Quantisierungsstrategie, die die spezifischen Charakteristiken dieses Rauschens nachbildet. Aus diesem Grund setzen wir eine stochastische Quantisierungsfunktion ein, die kontrollierbare Rauschstatistiken induzieren kann und gleichzeitig zu dünnbesetzten Matrizen im Trainingsalgorithmus führt. Diese Dünnbesetzung kann von effizienten Matrixmultiplikationsalgorithmen ausgenutzt werden um die erforderliche Anzahl an Operationen, und damit auch den Rechenaufwand, des Trainingsalgorithmuses zu verringern. Wir beweisen, dass, für gewisse Arten von neuronalen Netzen, gleichförmige Quantisierung mit additivem Störsignal die gewünschten Eigenschaften von kontrollierbarem Quantisierungsrauschen und hoher Dünnbesetzung erzielt. Experimente mit gängigen Bildklassifikationsproblemen bestätigen die theoretischen Eigenschaften unserer Methode und zeigen außerdem, dass die Methode Ansätze mit deterministischen Quantisierungsstrategien übertrifft, indem sie bessere Verhältnisse zwischen Generalisierung und Effizienz erzielt.

Acknowledgements

First, I want to thank Prof. Klaus-Robert Müller for handing me the topic of this thesis and for being the primary reviewer. Additionally, I want to thank Prof. Thomas Wiegand for also taking the time to review my thesis.

At Fraunhofer HHI, I want to thank Dr. Wojciech Samek for supervising me throughout my time there and for always providing helpful advice, even at short notice. I especially want to extend my gratitude to Simon Wiedemann for his strong and dedicated support. His always positive and cheerful manner, comprehensive knowledge of the field, and exceptional commitment were an indispensable help. This was invaluable for overcoming many obstacles and difficulties.

Finally, I want to thank Jan Laermann, Luis Oala, Sören Becker, Felix Sattler, Max Kohlbrenner, Patrick Wagner, Mohamed Gaafar, Vignesh Srinivasan and Marcel Ackermann for their comments, fruitful discussions, and very enjoyable company at Fraunhofer HHI.

Contents

1 Introduction

Artificial neural networks are powerful machine learning systems for recognizing patterns in large amounts of data. They became very popular through recent successes in computer vision, language understanding and other areas of computer science [1]. For example, neural networks are used by digital assistants for speech recognition and natural language processing, by e-commerce websites to make personalized recommendations, and by self-driving cars to detect objects in their surroundings. Compared to traditional machine learning algorithms, neural networks can handle raw input data without the need for domain-specific feature engineering. However, in order to extract meaningful representations from raw data, neural networks usually require large amounts of data. As a result, the training procedure is computationally very expensive and requires a lot of energy. For example, training a single state of the art image recognition model consumes approximately as much energy as an average household per week. Therefore, in order to train these models on resource-constrained hardware more efficient algorithms have to be designed. A lot of research is dedicated to make the forward pass through the network, which means the computation of new predictions, more efficient, however, there is less research focusing on a more efficient backward pass, which is also involved during training. More efficient training routines would increase the versatility of neural networks and enable new types of applications such as on-device training for small-scale hardware devices as needed for the internet of things.

Neural networks are usually trained using gradient-based optimization methods. This means that first a loss function is defined, which describes the prediction error of the network. Then the derivatives of this loss function with respect to the network parameters - the parameter gradients - are evaluated for the available training data using the back propagation algorithm. Lastly, the network parameters are adjusted according to the gradient information such that the prediction error is reduced. This process is repeated until the prediction error stops decreasing. In order to increase the efficiency of the training procedure we can leverage quantization techniques that can be applied to reduce the computational cost of the arithmetic operations performed by the back propagation algorithm. Quantization maps values from an input set to a finite (typically smaller) output set and is often used to save memory, processing power, and communication costs in digital systems. When applied to back propagation quantization injects noise into the resulting gradients and might thus interfere with the training process. Nevertheless, it was shown that using noisy gradient estimates to optimize neural networks can improve the generalization of the resulting models. Upon reviewing recent studies regarding different stochastic training methods we conclude that these gradient estimates should be unbiased, which means that the noise is zero in expectation, and that their noise variance is crucial for the generalization performance. Existing approaches to using quantization during training use deterministic quantization functions without theoretical guarantees on the statistical properties of the gradient error. In this thesis, we introduce a novel way of leveraging stochastic quantization to increase the efficiency of the gradient computation while maintaining unbiased parameter gradients with controllable noise variance. Our research question is whether stochastic quantization provides an advantage over deterministic quantization

approaches and how the induced quantization noise impacts the training procedure.

More specifically, our idea is to leverage uniform quantization with an additional stochastic noise signal, called dither, to compress the pre-activation gradients, which are intermediate results of the back propagation algorithm. We can show that for certain network architectures this specific quantization scheme induces a high degree of sparsity in the quantized pre-activation gradients. This sparsity can then be exploited to reduce the computational cost of the matrix multiplication operations in the back propagation algorithm, which are responsible for up to 90% of the overall computation time of the algorithm [2]. Since the pre-activation gradients are involved in two of three of these matrix multiplications, our method can achieve high computational savings during training. Furthermore, we will show that applying dithered uniform quantization to the pre-activation gradients does indeed lead to the desired unbiased parameter gradients with controllable noise variance.

The outline of this thesis is followed by chapter 2 where we discuss relevant background knowledge such as the back propagation algorithm for neural networks, how sparse matrix multiplication can decrease computational cost, and the implications of using noisy gradients for training neural networks. We then explore related work in the field of efficient neural network training and how these approaches relate to our method in chapter 3. Chapters 4 and 5 introduce dithered uniform quantization and how we can leverage this stochastic quantization function in conjunction with sparse matrix multiplication to reduce the complexity of the back propagation algorithm while fulfilling the desired statistical properties of the computed gradients. The experiments that we conduct to verify these properties and compare our method to related approaches are described in chapter 6. Finally in chapters 7 and 8 we summarize the work presented in this thesis, discuss the implications of our findings, and present future directions of research.

2 Background

In this chapter, we discuss the different concepts required as background knowledge for understanding Quantized Back Propagation (QBP) and the considerations behind it. The topics presented here do not necessarily have an immediate connection but will be unified later in chapter 5. We start with a description of the machine learning field in general and how neural networks are trained using the back propagation algorithm. Next, we show how matrix multiplications can be accelerated for sparse matrices with many zero-valued entries. This is important because matrix multiplications make up most of the computations during back propagation. In QBP matrix sparsity is obtained by stochastic quantization. However, quantization injects noise into the calculations and thus also into the result of the algorithm - the parameter gradients. At the end of this chapter we will discuss how noise contained in the parameter gradients affects the training procedure of neural networks.

2.1 Neural Networks and Deep Learning

In machine learning, a field of artificial intelligence, the goal is to design and apply algorithms that can "learn" to solve specific tasks based on data - without being explicitly programmed how to solve them. These algorithms build models by recognizing patterns in sample inputs that can then be used to make predictions for unseen data. We can distinguish between supervised and unsupervised learning scenarios. Supervised learning requires a "teacher" that presents sample inputs with their desired outputs, called labels, and the algorithm is supposed to find the general input-output mapping. Applications of supervised learning are classification tasks where inputs are assigned one or multiple classes or regression tasks where inputs are assigned continuous values. In unsupervised learning there are no labels and the algorithm is left to find structure in the input data. An example application of unsupervised learning is clustering where the inputs are assigned to groups which are not known beforehand. Artificial neural networks, or simply neural networks, are biologically inspired machine learning systems that have become very popular recently under the term deep learning [1]. The origins of artificial neural networks go back as early as 1949 to Hebbian learning [3], which mimics the adaption of neurons in the human brain during learning, and to the perceptron in 1958 [4], which is essentially a one-layered neural network that is applied to binary classification tasks. A neural network is a graph structure with nodes, which are referred to as neurons, that are arranged in multiple layers and which exhibit real-valued numbers. Each connection, also called synapse, weights the signal exhibited by a neuron and transmits it to a higher-level neuron. The neuron activation is computed by applying a function to the sum of its inputs, i.e. to the weighted sum of outputs of lower-level neurons. This activation function is usually non-linear and the same for all neurons in the network, except for the neurons in the last layer. The first layer of neurons represents the inputs to the system, the last layer represents the system output and the layers in between are called hidden layers and compute intermediate representations of the data. By adjusting the weights of the connections, the network can be "trained" to

produce desired output values. The structure of a fully-connected feed-forward neural network, i.e. an acyclic graph with all units in one layer being connected to all units in the next layer, is illustrated in figure 2.1. It was proven by the universal function

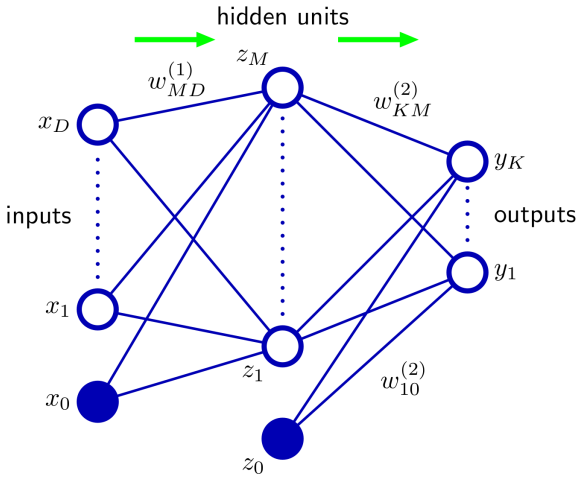

Figure 2.1: Fully-connected feed-forward neural network [5]

approximation theorem [6] that a feed-forward neural network with at least one hidden layer can approximate every continuous function in the Euclidean space. However, the theorem does not consider the trainabilty of such a network. In modern applications, neural networks with many hidden layers - deep neural networks - are found to perform better than shallow networks. For example, ResNets [7], which are among the most successful network architectures for image recognition tasks nowadays, can include over one hundred layers. There are different types of neural network architectures that are usually applied depending on the type of input and output data. For example, convolutional neural networks (CNNs), such as the aforementioned ResNets, are feed-forward networks that implement convolution and pooling operations, and which are most commonly applied to visual input for image or video recognition tasks because of their weight-sharing and translational invariance properties. Recurrent neural networks (RNNs) are cyclic networks that use an internal state to process temporal data which are thus often applied to problems with sequential inputs such as text or speech. Different network types can also be combined. For example, CNNs often use convolutional layers first and then fully-connected layers towards the end of the network. Also, in automated image captioning a CNN is combined with a RNN in order to process visual data and output text [8]. For the scope of this work however we will explicitly focus on fully-connected layers.

In neural networks, like in most machine learning scenarios, we consider two phases of the algorithm: Training the model and using it to make predictions. In the following sections we will go into more details about both phases.

2.1.1 Making Predictions

In a fully-connected neural network, each neuron computes a weighted sum of the outputs of all neurons from the previous layer. Then, to obtain the neuron's output - its activation value - an activation function $f(\cdot)$ is applied to the result of the sum.

Thus, for neuron i in layer l of the neural network the so called pre-activation and activation values of the neuron are defined as

$$z_i^l = \sum_{j=1}^{n_{l-1}} W_{ij}^l a_j^{l-1} + b_i^l, \tag{2.1}$$

$$a_i^l = f(z_i^l). \tag{2.2}$$

Here, z^l is the vector of pre-activation values and a^l is the vector of activation values in layer l with $l = 0$ being the inputs to the network, i.e. $a_i^0 = x_i$. Nowadays, the rectified linear unit (ReLU) [9]

$$f(x) = max(0, x) \tag{2.3}$$

is the most popular activation function [1]. The bias unit b_i^l can be thought of as an additional neuron in the previous layer that always emits the value 1 and that has an individual weight. The matrix $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ represents the weights of the individual connections between neurons in layer l and layer $l-1$ where n_l is the number of neurons in layer l. The weights and biases are the parameters of the model and get adjusted during training phase. To make predictions an input vector x is propagated through the network according to the equations given above until the output layer $l = L$ is reached. The network output is then given by $a^L = \hat{y}(x, \theta) = \hat{y}$ where θ represents the vector of all network parameters, i.e. weights and biases.

Notation remark. Throughout this thesis we use index l for the L layers of the network, index i for the n_l neurons in the respective layer, index j for the n_{l-1} neurons in the previous layer, and index h for the n_{l+1} neurons in the next layer. Thus, W_{ij}^l describes the weight between neuron i in layer l and neuron j in layer $l-1$. We use indices i', j', l' when iterating over other values of i, j or l respectively. Furthermore, as we will see in the next subsection, we use index k when iterating over training samples.

2.1.2 Training the Network

To train a neural network we first have to define a loss function, also called cost or error function. During training this loss function is evaluated based on the training data and minimized by adjusting the network parameters. This process is repeated for a desired number of iterations or until the computed loss is below a desired threshold. The loss function is usually chosen based on the task that is to be solved and evaluated based on a batch of training samples. For example, in regression, a squared, or quadratic, loss function can be used, which measures the difference between the network output and the label values:

$$l(y, \hat{y}) = \sum_{i=1}^{n_L} (y_i - \hat{y}_i)^2. \tag{2.4}$$

In classification, often the cross entropy loss function is used in conjunction with a softmax [10] activation function in the last network layer. The softmax function is a generalized logistic function where all entries are normalized such that they sum to one:

$$a_i^L = \frac{\exp z_i^L}{\sum_{i'} \exp z_{i'}^L}. \tag{2.5}$$

It is different from other activations function in that it is applied over the whole vector of pre-activations and not to every value individually. As a result, the last network layer outputs a probability distribution over the available classes. The cross entropy loss function [10] then measures the difference between the predicted distribution and the desired class distribution, which is usually a vector of ones and zeros:

$$l(y, \hat{y}) = -\sum_{i=1}^{n_L} y_i \log(\hat{y}_i). \tag{2.6}$$

The loss function is then evaluated for every training sample and the result is averaged:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{k=1}^{N} l(y^{(k)}, \hat{y}(x^{(k)}, \theta)). \tag{2.7}$$

Neural networks are usually trained with gradient-based methods. This means the loss function is minimized by evaluating its derivative with respect to the network parameters, i.e. its gradient, and adjusting the parameter accordingly. We also call this gradient the *parameter gradient* to distinguish it from gradients of the loss function with respect to other variables. For gradient-based methods all activation functions and the loss function have to be differentiable. To efficiently evaluate the loss gradient the back propagation algorithm applied, which we will discuss in more detail in the next section. Gradient descent algorithms are then used to adjust the network parameters according to the computed derivatives and a given learning rate:

$$\theta_i \longleftarrow \theta_i - \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta_i}. \tag{2.8}$$

In regular gradient descent (GD), one would evaluate the loss function over all training samples, as we defined it in equation 2.7, and use the resulting gradient to adjust the network parameters. However, for most modern applications with large data sets this is unfeasible. Therefore, stochastic gradients descent (SGD) is used, which estimates the loss function and its gradient based on a relatively small "batch" of training samples.

$$\widetilde{\mathcal{L}}(\theta) = \frac{1}{m} \sum_{k=1}^{m} l(y^{(k)}, \hat{y}(x^{(k)}, \theta)) \tag{2.9}$$

SGD variants therefor work very well for a wide variety of deep learning problems and are used by most practitioners [1, 11, 12].

Notation remark. From now on index k denotes the current sample from a batch with m training samples.

2.2 Back Propagation

Back propagation [13] is an algorithm to efficiently evaluate the derivatives of the loss function with respect to the network parameters for the given input data. Since neural networks apply elementary arithmetic operations and activation functions sequentially, the chain rule can be applied repeatedly in order to calculate the corresponding derivatives. Back propagation is a special case of automatic differentiation and is thus faster

and more accurate than symbolic or numerical differentiation. In neural networks, the term *back propagation of errors* is sometimes used because the error, which is calculated from the network output using the loss function, is propagated through the network in order to evaluate the partial derivatives of the loss function with respect to the parameters in the individual layers. The back propagation algorithm consists of three steps.

1. Forward propagation of the inputs through the network to obtain the outputs (forward pass).
2. Calculation of the error using the loss function.
3. Backward propagation of the error through the network to obtain the partial derivatives (backward pass).

So far we have discussed how to make predictions, i.e. to forward propagate input data through the network, and what loss functions can be used to evaluate the prediction error. To efficiently evaluate the gradient, the back propagation algorithm leverages principles from dynamic programming. During the backward pass the algorithm iterates through the layers from the last to the first layer. In each layer, the cached pre-activation and activation values that were computed in the forward pass are then used to compute the local error for the neurons. To better understand how the local errors are computed in each layer and before introducing a vectorized implementation of this algorithm, we first derive the derivatives of the loss function symbolically.

2.2.1 Partial Derivatives

To calculate the partial derivative of the loss function with respect to the weights of a fully-connected neural network, we first apply the chain rule to expand the calculation of the loss derivative with respect to a single weight:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = \frac{\partial \mathcal{L}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial W_{ij}^l}. \tag{2.10}$$

The derivative of the pre-activation with respect to the weights and the derivative of the activation with respect to the pre-activation are straightforward:

$$\frac{\partial z_i^l}{\partial W_{ij}^l} = \frac{\partial}{\partial W_{ij}^l} (\sum_{j'} W_{ij'}^l a_{j'}^{l-1} + b_i^l) = a_j^{l-1} \tag{2.11}$$

and

$$\frac{\partial a_i^l}{\partial z_i^l} = \frac{\partial f(z_i^l)}{\partial z_i^l} = f'(z_i^l). \tag{2.12}$$

For the last layer, the derivative is determined by the choice of loss function:

$$\frac{\partial \mathcal{L}}{\partial a_i^l} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \quad \text{for} \quad l = L. \tag{2.13}$$

For a lower layer, however, the derivative is less obvious. Since neurons in layer $l+1$, which all receive input from neuron i in layer l, are inputs to $\mathcal{L}$, we need to take the

total derivative, which is the sum of all partial derivatives with respect to the individual inputs, i.e. to all neurons in layer $l+1$:

$$\frac{\partial \mathcal{L}}{\partial a_i^l} = \sum_h \frac{\partial \mathcal{L}}{\partial a_h^{l+1}} \frac{\partial a_h^{l+1}}{\partial z_h^{l+1}} \frac{\partial z_h^{l+1}}{\partial a_i^l} \quad \text{for} \quad l < L. \tag{2.14}$$

We can now formulate the derivative of the loss with respect to a single weight in terms of "local errors" δ which are computed recursively:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = \delta_i^l a_j^{l-1} \tag{2.15}$$

with

$$\delta_i^l = \frac{\partial \mathcal{L}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} = \begin{cases} \dfrac{\partial \mathcal{L}}{\partial \hat{y}} f'(z_i^l), & \text{if } l = L \\ \displaystyle\sum_h \delta_h^{l+1} W_{hi}^{l+1} f'(z_i^l), & \text{otherwise.} \end{cases} \tag{2.16}$$

which represents the propagation of errors through the network. The partial derivative of the loss function with respect to the bias is simply

$$\frac{\partial \mathcal{L}}{\partial b_i^l} = \frac{\partial \mathcal{L}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_i^l} = \delta_i^l. \tag{2.17}$$

In the next subsection, we describe how the forward pass and the backward pass, i.e. the evaluation of these derivatives, can be vectorized and implemented layer-wise.

2.2.2 Vectorization

To describe the implementation of the back propagation algorithm, we first express the forward and backward pass in terms of vectors and matrices. We slightly change our notation to accommodate the evaluation of the loss function and its gradient for multiple data samples in one step. We refer to the the size of the batch of training samples that are propagated concurrently as m. Then, the input to the network is not a vector anymore but a matrix $x \in \mathbb{R}^{0 \times m}$. Therefore, $z^l, a^l \in \mathbb{R}^{n_l \times m}$ represent the pre-activation and activation matrices. The activation function $f(\cdot)$ is applied to every element of the matrix independently. We denote the matrix of partial derivatives of the loss function with respect to the pre-activation and activation values evaluated for the single input samples as follows:

$$\frac{\partial \mathcal{L}}{\partial z_{ij}^l} = dz_{ij}^l, \quad \frac{\partial \mathcal{L}}{\partial a_{ij}^l} = da_{ij}^l. \tag{2.18}$$

We will refer to these matrices as the *pre-activation* and *activation gradients*. Similarly, the *weight* and *bias gradients* are given by

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = dW_{ij}^l, \quad \frac{\partial \mathcal{L}}{\partial b_{ij}^l} = db_{ij}^l. \tag{2.19}$$

Using this new notation, the vectorized forward pass is defined as

$$z^l = W^l a^{l-1} + b^l \ \mathbf{1}^T, \tag{2.20}$$

$$a^l = f(z^l) \tag{2.21}$$

and correspondingly, the vectorized backward pass is given by

$$dz^l = da^l \odot f'(z^l), \tag{2.22}$$

$$dW^l = dz^l (a^{l-1})^T, \tag{2.23}$$

$$db^l = dz^l \ \mathbf{1}, \tag{2.24}$$

$$da^{l-1} = (W^l)^T dz^l. \tag{2.25}$$

with $\odot$ denoting the element-wise product, and the one-vector $\mathbf{1} \in \mathbb{1}^m$. Note that the local errors δ^l that we described in the last subsection are represented here as the pre-activation gradients dz^l.

As we can see, the forward and backward pass can be computed using only element-wise addition and multiplication, matrix-vector and matrix-matrix multiplications, and element-wise application of the activation function. There are many software libraries for the efficient implementation of these linear algebra routines which can be leveraged for training neural networks. For example, Basic Linear Algebra Subroutines (BLAS) [14] is a collection of common linear algebra operations with multiple implementations which are leveraged in higher-level frameworks. Most software frameworks for deep learning, such as Caffe [15], PyTorch [16] and TensorFlow [17] build on these implementations to expose programming interfaces for training neural networks.

2.2.3 Computational Cost

Asymptotically, the total number of operations involved in the back propagation algorithm is dominated by the matrix multiplication operation. All other arithmetic operations are element-wise addition or multiplication. Thus, we approximate the cost of back propagation by the number of operations involved in the matrix multiplications. In this work, we focus on reducing this number of operations in order to reduce the cost of training neural networks.

Generally, when multiplying a matrix A with a vector x the values in the output vector y are obtained by computing inner products between the row vectors in A and x:

$$y_i = \sum_j A_{ij} x_j \tag{2.26}$$

As a result, for a matrix-vector multiplication between $A \in \mathcal{R}^{m \times n}$ and $x \in \mathcal{R}^n$ there are mn multiplication and accumulation (MAC) operations needed. Matrix-vector multiplication can easily be extended to matrix-matrix multiplication because for the latter multiple matrix-vector multiplications have to be performed, namely as many as the right matrix has columns:

$$C_{ij} = \sum_k A_{ik} B_{kj} \tag{2.27}$$

Therefore, a matrix-matrix multiplication between $A \in \mathcal{R}^{m \times n}$ and $B \in \mathcal{R}^{n \times p}$ requires $\mathcal{O}(mnp)$ MAC operations.

Since all of the three matrix multiplications in the forward and backward pass (see equations 2.20, 2.23 and 2.25) have the same input dimensionalities, we can denote the cost of back propagation as

$$E_{BP} = \mathcal{O}(n_l n_{l-1} m). \tag{2.28}$$

In the next section we will discuss how some of these MAC operations can be omitted by exploiting the sparsity of the involved matrices.

In order to estimate the energy cost of training neural networks one can compare the power usage of a modern hardware accelerator and the training time of a state-of-the-art image recognition model. The popular NVIDIA Tesla P100 GPU accelerator consumes up to 250W[1], ignoring the memory consumption of other necessary hardware like CPU, main memory, etc. Training a ResNet on a large image dataset for object recognition takes either around 29 hours on eight such accelerators or one hour on 256 accelerators [18], which both results in an energy consumption of approximately 60 kWh. As an interesting fact, the energy cost of this training procedure roughly corresponds to a household's weekly energy budget, assuming a yearly energy consumption of around 3000 kWh for average housholds in Germany[2].

2.3 Sparse Matrix Multiplication

As described before, in this work we want to explore how to reduce the computational cost of training neural networks. So far, we have described the training procedure and the back propagation algorithm and its complexity. In order to reduce this complexity QBP applies a quantization scheme that sparsifies the pre-activation gradients in the backward pass before they are used to perform the matrix multiplications defined in equations 2.23 and 2.25. In this section, we will discuss how one can exploit this sparsity property to reduce the computational cost of matrix multiplications. The reduced cost of the matrix multiplications then leads to a overall reduced cost of the back propagation algorithm. The memory formats and algorithms presented here are taken from the source code of SciPy [19], a popular open-source software library for scientific computing.

As discussed in the previous section, multiplying the matrices $A \in \mathcal{R}^{m \times n}$ and $B \in \mathcal{R}^{n \times p}$ requires $\mathcal{O}(mnp)$ MAC operations. If one of the matrices is known to be sparse, i.e. exhibits a high ratio of zero-valued elements, the matrix multiplication can be sped up. Obviously, we do not need to perform operations on matrix elements that are zero. Therefore, when multiplying a sparse matrix $S \in \mathcal{R}^{m \times n}$ with $||S||_0$ non-zero elements and a dense matrix $B \in \mathcal{R}^{n \times p}$, we only need to perform $||S||_0 \, p$ MAC operations. Usually, matrices are represented in memory as two-dimensional arrays and are then traversed sequentially by the naive matrix multiplication algorithm. As a result, the algorithm would iterate through all elements in the sparse matrix and thus not achieve reduced time complexity. In order to only iterate and operate on non-zero entries we first need to convert the sparse matrix to a different memory format. There are many memory formats for sparse matrices such as dictionary of keys (DOK),

[1] https://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf

[2] https://www.destatis.de/DE/ZahlenFakten/GesamtwirtschaftUmwelt/Umwelt/UmweltoekonomischeGesamtrechnungen/MaterialEnergiefluesse/Tabellen/StromverbrauchHaushalte.html

lists of lists (LIL), coordinate list (COO), compressed sparse row (CSR), compressed sparse column (CSC), etc. They all have advantages and disadvantages. For example, DOK, LIL and COO are easier to construct but CSR and CSC allow for faster matrix-vector products. The COO format is the most straightforward one because it is just a coordinate representation of the non-zero elements. It contains three arrays with the same length: Row, column and data. The row and column arrays represent the row-column coordinate tuples of the non-zero elements and the data array represents the corresponding values. A COO matrix is easy to construct because the source matrix just has to be iterated while filling the three arrays for every non-zero element. However, the CSR and CSC formats are more suitable for efficient matrix multiplication algorithms. These formats are similar to COO but compress the row, or column, array for faster row, or column, access.

For the CSR format, the data and column array are kept as in COO. The row array will be compressed - hence the name - such that it stores the number of non-zero elements in the single rows cumulatively. More specifically, the first entry in the row array is $\boldsymbol{r}[0] = 0$. Then, $\boldsymbol{r}[i+1]$ stores the position of the first non-zero element of row i in the data array $\boldsymbol{s}$. As a result, the difference between adjacent entries in the row array represent the number of non-zero elements in each matrix row. For example, for the matrix

$$S = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 3 \end{bmatrix}$$

the three arrays would be given by

$$\begin{aligned} \text{data} &= [1, 2, 3] \\ \text{column} &= [1, 0, 2] \\ \text{row} &= [0, 1, 1, 3]. \end{aligned}$$

Now, one can efficiently compute matrix-vector products by iterating through the row array, as described in algorithm 1. Figure 2.2 illustrates the computations of this

input : CSR matrix $S \in \mathcal{R}^{m \times n}$ with data, row and column arrays $\boldsymbol{s}$, $\boldsymbol{r}$ and $\boldsymbol{c}$, and vector $v \in \mathcal{R}^n$ with data array $\boldsymbol{v}$
output: vector $y = Sv$

```
for i ← 0 to m − 1 do
    v[i] ← 0;
    for j ← r[i] to r[i + 1] − 1 do
        v[i] ← v[i] + s[j]v[c[j]];
    end
end
```

Algorithm 1: Matrix-vector product for sparse matrix in CSR format and dense vector

algorithm. As we can see, using the compressed row format, for every row we only iterate through the columns of the sparse matrix that have non-zero elements. Since, in total, this is limited by the overall number of non-zero entries in the matrix, or the number of rows, the time complexity of this algorithm is $\mathcal{O}(||S||_0 + m)$.

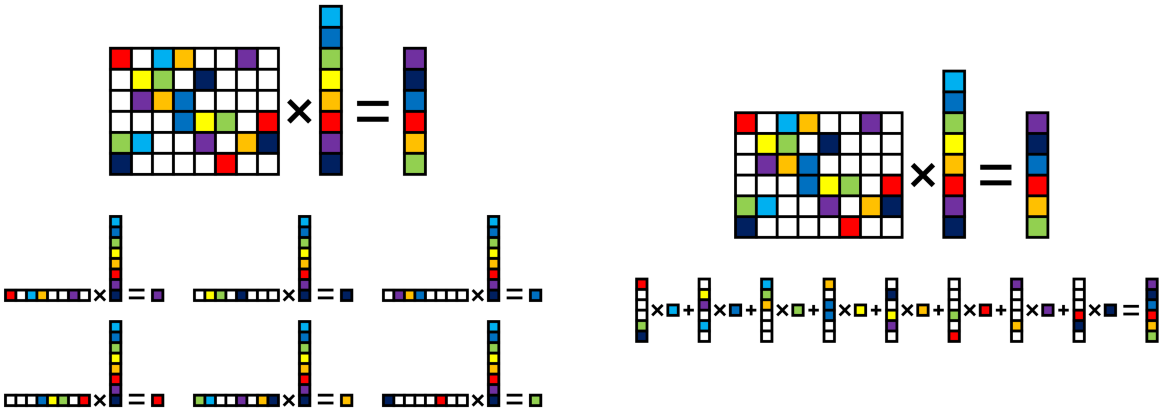

Figure 2.2: Multiplication of a CSR matrix (left) or a CSC matrix (right) with a dense vector. Figures from [20].

When we include the construction of the sparse memory format, which has complexity $\mathcal{O}(mn)$, then there is no advantage in performing the sparse matrix multiplication over a regular matrix multiplication. However, when we extend this algorithm to multiplication of a sparse matrix and a dense matrix, the complexity can be lower than for the naive matrix-matrix multiplication algorithm. We can extend the algorithm from above to multiplying the sparse matrix S with a dense matrix $B \in \mathcal{R}^{n \times p}$ by replacing the instruction in the inner loop with another loop over the p columns of B to compute a row of the result matrix using the non-zero elements of the current row of S. The run-time of this adapted algorithm is

$$E_{CSR}(SB) = \mathcal{O}(||S||_0 p + mp), \tag{2.29}$$

where mp results from the fact that we need to visit every output element once in order to at least assign it the zero value. We can also adapt this algorithm to when S is formatted using CSC which results in a slightly worse theoretical run-time of

$$E_{CSC}(SB) = \mathcal{O}(||S||_0 p + mp + n). \tag{2.30}$$

The additional n summand comes from the fact that we have to iterate over the columns of the CSC matrix. Algorithm 1 can also be adapted to the case where the sparse matrix is the right argument of the matrix multiplication. The complexity for multiplying a dense matrix $C \in \mathcal{R}^{q \times m}$ with S is given as

$$E_{CSC}(CS) = \mathcal{O}(||S||_0 q + nq) \tag{2.31}$$

given S is formatted as CSC, and

$$E_{CSR}(CS) = \mathcal{O}(||S||_0 q + nq + m) \tag{2.32}$$

for the CSR memory format. In this case, CSC is the better choice. We will see in chapter 5 how the computational benefits of sparse matrix multiplication can be applied to the back propagation algorithm.

2.4 Noisy Parameter Gradients

So far we have described how to train neural networks and shown that the back propagation algorithm for fully-connected networks relies heavily on matrix multiplications. In the last section, we have also discussed how sparse matrices can reduce the computational cost of matrix multiplications. In order to leverage this computational benefit we can make use of a sparsifying quantization scheme during the back propagation algorithm, as we will describe in chapter 5. We will use a stochastic quantization function with controllable statistical properties. However, stochastic quantization applied to the back propagation algorithm introduces noise into the resulting parameter gradients. In this section we discuss the general implications of using noisy parameter gradients for training neural networks.

One could argue that the noise in the gradient signal might hinder the gradient-based optimization. The noisy gradients do not represent the full information of the original gradient directions. Thus, the gradient-based training procedure might take longer to converge and potentially converge to worse solutions. Despite this, gradient noise can also be beneficial to the optimization procedure and the resulting network configurations. The most prominent example for training with noisy gradients is stochastic gradient descent. As discussed in subsection 2.1.2, it evaluates the gradient based on a subset of the training data. This results in a noisy estimate of the true gradient computed by regular gradient descent. The gradient estimate is unbiased since in expectation it equals to the true gradient.

$$\nabla\widetilde{\mathcal{L}} = \nabla\mathcal{L} + \epsilon \quad \text{with} \quad \mathbb{E}[\epsilon] = 0 \tag{2.33}$$

While the main motivation for using SGD over GD for training neural networks is its feasibility for large data sets, it was also shown that SGD converges faster and to better solutions [21]. Note that in machine learning the training procedure minimizes the prediction error on the training samples - the empirical loss - while the quality of the resulting solution is measured by its prediction error on new data samples - its generalization performance. It is not entirely clear why solutions found by SGD lead to better generalization performance than GD. There is however evidence that the noise characteristics in the parameter gradients play an important role.

Training neural networks corresponds to finding local minima in a highly non-convex optimization landscape. It was shown that noise in SGD can help to escape saddle points [22, 23] and lead to local minima that generalize better [24, 25]. One hypothesis to why SGD finds these superior minima is that the noise drives the optimization away from "sharp" local minima and towards "flat", or "broad", minima [21, 26, 27]. Flat minima for which the eigenvalues of the Hessian matrix of the loss function are small are conjectured to generalize better [21, 28–30]. For flat minima, small changes in the network configuration do not result in a drastically different empirical loss. Thus, the intuition is that these solutions correspond to models that overfit less to the training data. This means that instead of memorizing the specifics of the training data they recognized patterns that can be used to achieve good generalization on new data. The hypothesis of gradient noise leading to flat minima which generalize better would also explain why deeper over-parameterized networks with increased model complexity do not necessarily lead to more overfitting [26], which is what classical learning theory would predict. Since the gradient in SGD drives the optimization towards deep minima

and the noise drives it towards flat minima, one could say that the amount of noise influences the speed of the optimization and the quality of the solutions. Recently, it has been proposed that SGD has a general "noise scale" that is approximately $\eta\, N/B$ where η is the learning rate, N is the number of training samples and B is the batch size [31]. Increasing the batch size leads to a better estimation of the original gradient and thus to less noise. A higher learning rate, however, scales up the noise contained in the gradients and thus leads to a larger overall noise scale. This matches with the general knowledge that the choice of learning rate and batch size are important hyperparameters for finding solutions that generalize well, and also coincides with findings that the right learning rate schedule is crucial for escaping bad local minima [32]. They also propose that due to the central limit theorem the gradient noise induced by SGD is unbiased Gaussian noise with unknown variance.

So far we have established that the noise contained in the gradient estimates computed by SGD might be the reason for the good generalization performance of its solutions and thus its success in training neural networks. Additionally, there have been findings that adding artificial Gaussian noise to the parameter gradients computed by SGD, which increases the noise scale, can further improve the generalization performance of the resulting models [26, 33, 34]. Also, gradient noise can be used to view stochastic gradient-based optimization as a form of approximate Bayesian posterior sampling. For example, Stochastic Gradient Langevin Dynamics (SGLD) [33] describes a way of adding unbiased Gaussian noise with decaying variance to the parameter gradients such that gradient iterates converge to samples from the true posterior over the network parameters. This posterior distribution captures the uncertainty over the parameters and thus sampling from it yields models that are less prone to overfitting. However, for highly non-convex optimization landscapes, such as for neural networks, adding uncorrelated Gaussian noise does not reflect the different noise sensitivities of the individual parameters. Thus, the variance of the added noise has to be pre-conditioned to the curvature of the optimization landscape [35, 36]. These Bayesian approaches of approximating a posterior distribution over the model parameters have not yet outperformed simpler noise-based regularization heuristics such as dropout [37], which also lead to solutions with better generalization performance. However, Bayesian inference offers useful tools for the theoretical analysis of the role of noise in stochastic optimization. Also, the additional noise described by SGLD was applied in a non-Bayesian fashion to regular SGD and was shown to sometimes increase the stability of training complex neural networks [34]. We can conclude again that unbiased Gaussian noise with appropriate variance is helpful to converging to networks that generalize well.

In conclusion, we conjecture that SGD is so successful in training neural networks that generalize well because it induces unbiased Gaussian noise with appropriate preconditioning. This means, it adapts the noise variance of the parameters in a way that's beneficial for the convergence to good solutions. Since our goal is to use quantization during back propagation, which also adds noise the the gradients, in this work, we are looking for a quantization scheme that induces gradient noise which exhibits similar properties as the noise induced by SGD. Therefore, our proposed method produces unbiased gradient estimates that have controllable variance. As a result, the method can be adjusted to achieve the optimal noise scale and thus the best trade-off between compression and generalization performance.

3 Related Work

Surprisingly, there is not a lot of previous work on more efficient neural network training. One approach to this problem is precision quantization [38–51]. Here, quantization is used to transform activation, weight, and gradient values from a regular single-precision floating point format to different compressed representations. In these new representations, multiplications and addition operations can be conducted more efficiently. For example, [38] were one of the first to use a low-bit fixed-point format with dynamic scaling to represent the variables in the back propagation algorithm. Later approaches even used binary weights and activations to replace arithmetic with logical operations [40–42]. However, the gradient values still need to be represented with high precision. This is improved in [44, 45] by showing that also low-bit gradient representations are feasible. Most of the approaches reviewed so far cannot be applied to commodity hardware. The authors propose custom hardware designs to leverage the low-bit representations for efficient computation. Furthermore, the proposed quantization heuristics are not theoretically motivated and only verified experimentally. It is not clear how the induced error on the weight gradients is defined and how it impacts the generalization performance.

In contrast, [52] and [53] propose stochastic quantization techniques for efficient communication in distributed training and also provide theoretical convergence guarantees. They leverage a sparsifying quantization scheme that is unbiased and leads to a bounded norm of the gradient vectors. Both approaches first compute the full-precision parameter gradients before quantization. Our work, however, focuses on the efficient approximation of the parameter gradients in order to save computation and is thus orthogonal to distributed scenarios.

We found two studies that, similar to our work, explore the use of efficient approximations to the matrix multiplications in the back propagation algorithm. [54] shrink the relevant matrices by column-row-sampling [55] or a heuristic variation thereof. Column-row-sampling selects an optimal subset of column-row pairs from two matrices such that the matrix multiplication of the shrunken matrices is an unbiased estimation of the original matrix product with bounded error norm. Using an efficient sampling heuristic, this approach achieves up to 80% reduced computation but the authors provide no analysis of the induced noise variance contained the weight gradients and its impact on the generalization performance. The group sparsity induced by column-row-sampling during training is a special case of our method of using quantization to induce sparsity on all matrix elements. The meProp algorithm [2] is the work most closely related to ours and describes how to sparsify the pre-activation gradients and leverage sparse matrix multiplications for a more efficient backward pass. Their *top-k* quantization algorithm selects the k elements with the largest magnitude from every pre-activation gradient vector that corresponds to a single batch sample. Since this quantization function is deterministic and operates on vectors, it results in a biased quantization error that is correlated among the vector elements and has no theoretical bounds. In this thesis, we show that a stochastic quantization function applied element-wise to the pre-activation gradients can overcome these limitations, describe statistical properties of the method error and achieve better generalization-compression ratios than meProp.

4 Dithered Uniform Quantization

In the last two chapters we discussed the background topics for our goal of increasing the efficiency of neural network training and existing approaches to this problem. For Quantized Back Propagation our goal is to develop a method that computes computationally cheaper parameter gradient approximations that still lead to good solutions. As discussed in section 2.4, we want our method to produce gradient estimates that are unbiased and have controllable noise variance. As we will show in the next chapter, in order to fulfill these desired properties we require a stochastic quantization scheme that is unbiased, has controllable error variance, and uncorrelated quantization errors for distinct inputs. It also needs to induce sparsity, which can then be exploited to reduce the computational cost of subsequent sparse matrix multiplications. In this chapter we introduce a quantization method that fulfills all of the above requirements - dithered uniform quantization. The concepts discussed here are leveraged in chapter 5 for the design and analysis of the QBP algorithm. Note that in this chapter we rely heavily on early work by Schuchman [56] and reviews by Wannaker, Lipshitz and Vanderkooy [57–59] regarding the theoretical properties of using dither signals in uniform quantization. Before explaining the use of dither signals and their effects on the quantization error, we first introduce regular, *undithered* uniform quantization.

4.1 Uniform Quantization

Quantization can be defined as a surjective map from a set of real-valued, possibly continuous elements to another real-valued set which is finite and has lower cardinality. It forms the basis of data encoding, which partially discards information to obtain reduced size for data storing, handling and transmitting. Uniform quantization is a particular type of quantization function which was originally introduced for analog-to-digital conversions of audio signals. The function rounds a given value to its nearest quantization point. These quantization points are evenly distributed along the input dimension with a distance equal to the quantization step size. For instance, a *mid-tread* uniform quantization function is defined as follows[1]:

$$Q(x) = \Delta \lfloor \frac{x}{\Delta} + \frac{1}{2} \rfloor \tag{4.1}$$

where x is an input signal and Δ is the quantization step size. The quantization function can be separated into two stages. The classification stage, or also called forward quantization or encoding, which rounds the input value to the next quantization index:

$$i = \lfloor \frac{x}{\Delta} + \frac{1}{2} \rfloor. \tag{4.2}$$

And the reconstruction stage, or also called inverse quantization or decoding, that scales the quantization index by the step size:

$$y = \Delta i. \tag{4.3}$$

Regular uniform quantization and other deterministic quantization functions such as the top-k algorithm from [2] result in a deterministic quantization error

$$q(x) = Q(x) - x \tag{4.4}$$

that is fully dependent on the input signal. This means that there is a deterministic mapping from the distinct input signals to their corresponding quantization errors. As a result, for distinct inputs the quantization error is not zero in expectation and we can therefore say that the quantization function is biased. The left plot in figure 4.1 illustrates the deterministic quantization error of uniform quantization, which follows a sawtooth wave due to the rounding function. Although the quantization error is input-dependent, the classical model of quantization [60] states that with a small step size relative to the input distribution the bias in the quantization error can be neglected and the error can be modeled as form of additive, unbiased noise. However, when the quantization step size is larger, the error's input-dependence can result in a significant bias. Luckily, undithered uniform quantization, as we introduced it here, can be turned into a stochastic quantization function using a *dither* signal. As a result, one can analyze the statistical moments of the quantization error.

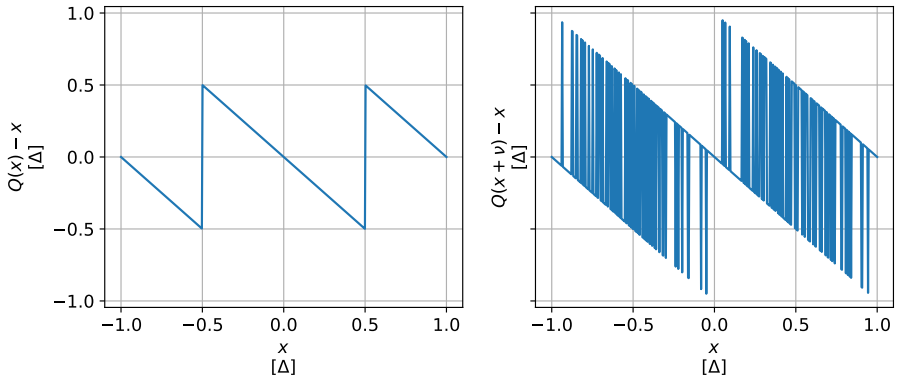

Figure 4.1: Quantization error for uniform quantization without dither signal (left) and with dither signal (right).

4.2 Non-Subtractive Dither

Dither describes a noise signal that is intentionally added to the input of the quantization function to randomize the induced quantization error. This is illustrated by the right plot in figure 4.1. Using a dither signal we can model the output and the total error of the quantization scheme as random variables and and describe their statistical moments. A uniform quantization system with additive dither signal, also called non-subtractive dither (NSD), is defined as follows:

$$y = Q(x + \nu) = \Delta \lfloor \frac{x + \nu}{\Delta} + \frac{1}{2} \rfloor \tag{4.5}$$

[1] In contrast, *mid-riser* uniform quantization is defined as $Q(x) = \Delta \lfloor x/\Delta \rfloor + 1/2$. Since this quantization function cannot produce zero-valued outputs and our goal is to obtain a sparsifying quantization scheme, we introduce mid-tread quantization here.

where the dither signal ν is drawn from a probability distribution with probability density function (pdf) p_ν. The total error, or quantization error, is defined as the difference of the system output and the system input:

$$\epsilon = y - x = Q(x + \nu) - x. \tag{4.6}$$

As we can see, the quantization error, is still dependent on the input. However, sampling the dither signal from an appropriate probability distribution, the statistical moments of the quantization error are decoupled from the input[2]. This allows us to describe these moments for arbitrary input distributions. One can think of uniform quantization with NSD as a one-dimensional grid with a randomly sampled offset for every incoming value which is then moved to the nearest grid node. In the remainder of this section we want to analyze the error moments induced by dithered uniform quantization using appropriate dither distributions.

4.2.1 Appropriate Dither Distributions

According to [57], in order to decouple the first m moments of the quantization error of a uniform quantization system with NSD from the system input, the dither signal has to be drawn from a distribution that fulfills the following condition:

$$\mathcal{P}_\nu^{(i)}\left(\frac{k}{\Delta}\right) = 0 \qquad \forall k \in \mathbb{Z}_0 \quad \text{and} \quad i = 0, 1, \ldots, m-1 \tag{4.7}$$

with Δ being the quantization step size, $\mathcal{P}_\nu$ the characteristic function (cf) of the dither signal, i.e. the Fourier transform of its pdf p_ν, and $\mathcal{P}_\nu^{(i)}$ its i-th derivative. Furthermore, it was shown that when the above condition is fulfilled for a particular value of m, the first m moments of the quantization error are given by

$$\mathbb{E}[\epsilon^m] = \sum_{l=0}^{\lfloor \frac{m}{2} \rfloor} \binom{m}{2l} \left(\frac{\Delta}{2}\right)^{2l} \frac{\mathbb{E}[\nu^{m-2l}]}{2l+1} \tag{4.8}$$

which results in the following equations for the first two moments:

$$\mathbb{E}[\epsilon] = \mathbb{E}[\nu], \tag{4.9}$$

$$\mathbb{E}[\epsilon^2] = \mathbb{E}[\nu^2] + \frac{\Delta^2}{12}. \tag{4.10}$$

As we can see, using an appropriate dither signal, the moments of the quantization error are directly determined by the moments of the dither signal. This allows us to control the error moments by the choice of dither signal. Moreover, for dither signals that fulfill the above condition for $m \geq \max(m_1, m_2)$, one can derive a general equation for the joint error moments $\mathbb{E}[\epsilon_1^{m_1}\epsilon_2^{m_2}]$, which is given in [57]. More importantly, if this dither signal is sampled independently the joint error moments are uncorrelated:

$$\mathbb{E}[\epsilon_1^{m_1}\epsilon_2^{m_2}] = \mathbb{E}[\epsilon_1^{m_1}]\mathbb{E}[\epsilon_2^{m_2}]. \tag{4.11}$$

[2] Alternatively, one could use subtractive dither (SD) where the quantization error and its moments are entirely decoupled from the input signal. However, since for SD the dither signal is subtracted again after decoding, the quantized values are almost never zero, which is unsuitable for a sparsifying quantization scheme.

Thus, using an independently sampled dither signal that fulfills the condition from above for $m \geq 1$, the quantization error of distinct input samples is uncorrelated:

$$\begin{aligned} \text{cov}(\epsilon_1, \epsilon_2) &= \mathbb{E}[\epsilon_1 \epsilon_2] - \mathbb{E}[\epsilon_1]\mathbb{E}[\epsilon_2] \\ &= \mathbb{E}^2[\epsilon] - \mathbb{E}^2[\epsilon] \\ &= 0. \end{aligned} \tag{4.12}$$

Recall the three properties that we defined in the beginning of the chapter for the desired stochastic quantization scheme: 1. Unbiasedness, 2. controllable error variance, and 3. uncorrelated error for distinct input samples. Using a dither signal that has a mean of zero, i.e. $\mathbb{E}[\nu] = 0$, and is sampled independently from a distribution that fulfills the condition in equation 4.7 for $m \geq 2$, we obtain a stochastic quantization function that fulfills all three requirements. Property 1 and 3 follow directly from equations 4.9 and 4.12. Property 2 follows from equation 4.10 and the fact that the variance is equal to the second moment in case the mean is zero. In the next subsection we introduce such a dither distribution.

4.2.2 Triangular Dither

We refer to a dither signal that is independently sampled from a triangular pdf with a mean of zero and an amplitude of 2Δ as *triangular dither*. It can be shown that using such a dither distribution fulfills the condition from equation 4.7 for $m = 2$ and thus renders the first two moments of the quantization error independent from the input signal. Furthermore, as shown by [59], it is the only dither signal that fulfills this condition while minimizing the second moment of the quantization error. Therefore, triangular dither is an optimal choice for input-independent error mean and variance. We can sample from such a triangular dither distribution by summing two independent random samples from a rectangular pdf, also called uniform distribution, each with a mean of zero and an amplitude of Δ.

$$\nu = \nu_1 + \nu_2 \quad \text{with} \quad \nu_1, \nu_2 \sim \mathcal{U}(-\frac{\Delta}{2}, \frac{\Delta}{2}). \tag{4.13}$$

This follows from the fact that the summation of two independent random processes convolves their pdfs and the convolution of the two rectangular pdfs mentioned here precisely results in the desired triangular pdf, as illustrated in figure 4.2. Since a uniform distribution $\mathcal{U}(-a, a)$ has second moment ${}^{(2a)^2}/_{12}$, the triangular dither signal has second moment

$$\mathbb{E}[\nu^2] = \mathbb{E}[\nu_1^2] + \mathbb{E}[\nu_2^2] = \frac{\Delta^2}{6} \tag{4.14}$$

and using equations 4.9 and 4.10 we can show that the resulting quantization error has first and second moment

$$\mathbb{E}[\epsilon] = 0, \tag{4.15}$$

$$\mathbb{E}[\epsilon^2] = \frac{\Delta^2}{4}. \tag{4.16}$$

It seems that we have found a good candidate for the dither distribution to use in a uniform quantization scheme that we can leverage later in our quantized back propagation algorithm. However, as we will see in the next subsections, there is an alternative choice of dither signal that, despite not satisfying all the desired properties, has some important advantages for the use in our method.

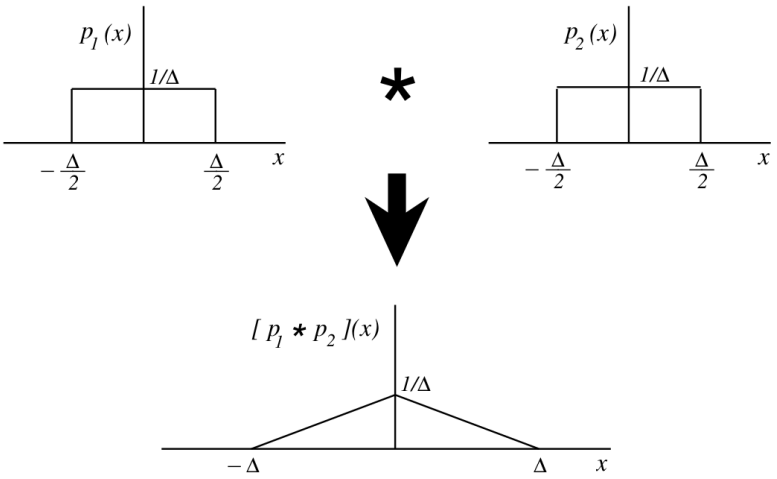

Figure 4.2: Convolution of two zero-centered uniform distributions with amplitude Δ. Figure from [59].

4.2.3 Uniform Dither

We refer to a dither signal that is independently sampled from a rectangular pdf with a mean of zero and an amplitude of Δ as *uniform dither*. This is precisely the distribution we sampled from twice to obtain the triangular dither signal in the last subsection. Uniform dither, as described here, fulfills the condition in equation 4.7 for $m = 1$ and thus only renders the first moment of the quantization error independent from the input. Therefore, the variance of the quantization error is still dependent on the input and thus not directly controllable by the choice of quantization step size, as it is the case for triangular dither. However, we found that the variance of the quantization error that results from using a uniform dither signal is bounded by the error variance that results from using triangular dither. We summarize this finding in the following conjecture which we verify experimentally in appendix B.1.

Conjecture. *Let Q be a mid-tread uniform quantization function with quantization step size Δ*

$$Q(x) = \Delta \lfloor \frac{x}{\Delta} + \frac{1}{2} \rfloor, \tag{4.17}$$

and ν a random variable that follows a uniform distribution with a mean of zero and amplitude equal to the quantization step size

$$\nu \sim \mathcal{U}(-\frac{\Delta}{2}, \frac{\Delta}{2}). \tag{4.18}$$

Then, the quantization error of a uniform quantization system with non-subtractive dither

$$\epsilon = Q(x + \nu) - x \tag{4.19}$$

has a second moment bounded by

$$\mathbb{E}[\epsilon^2] \leq \frac{\Delta^2}{4}. \tag{4.20}$$

As stated by the conjecture above, given the same quantization step size, the variance of the quantization error induced by uniform dither is less than or equal to the quantization error induced by triangular dither. Thus, applying uniform quantization to the back propagation algorithm, uniform dither introduces less or equally much noise variance to the training process. However, the noise that is actually induced has input-dependent variance - the noise is modulated. Instead of controlling the exact variance of the induced noise, we can only control the upper bound of the variance. This is a disadvantage since our desired quantization scheme should have controllable error variance such that the method can adapt to a required noise scale for the resulting parameter gradients, as motivated in section 2.4. However, input-dependent noise variance during the back propagation algorithm, could also represent a helpful pre-conditioning of the noise regarding the training procedure. As a result, it is not obvious whether the lower but modulated noise variance induced by using uniform dither will generally be helpful or not when applying this quantization scheme during the backward pass. Furthermore, not only the induced noise variance but also the induced sparsity is an important criterion of the quantization scheme. As it will become clearer in the next section for relevant input distributions uniform dither induces higher sparsity ratios but at the same time much less noise variance as compared to triangular dither.

4.3 Induced Sparsity

The output sparsity of a dithered uniform quantization scheme is dependent on the input and the dither distributions. Since, as we will see in the next chapter, the computational benefits of QBP are dependent on the output sparsity of the quantizer, this property is very important to our method. Thus in this section we want to analyze the induced sparsity in general and also for input distributions that are relevant to our method.

In order to analyze the output sparsity of a dithered uniform quantization system we first have to formalize the distribution of the values that are being quantized. These values consist of the system inputs and the dither samples:

$$s = x + \nu \tag{4.21}$$

$$y = Q(s) \tag{4.22}$$

To describe the sparsity of the output y we first need to define the pdf of s. Since, the value of s is given by the sum of two random variables, its pdf is the convolution of the summand pdfs:

$$\begin{aligned} f_s(s) &= (f_x * f_\nu)(s) \\ &= \int_{-\infty}^{\infty} f_x(s-\tau) f_\nu(\tau) d\tau \end{aligned} \tag{4.23}$$

where f_ν is the pdf of the dither signal which is defined as

$$f_{\nu_u}(t) = \begin{cases} \frac{1}{\Delta} & \text{if } -\frac{\Delta}{2} \leq t \leq \frac{\Delta}{2}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.24}$$

for uniform dither and as

$$f_{\nu_t}(t) = \begin{cases} \frac{1}{\Delta}(1 - \frac{|t|}{\Delta}) & \text{if } -\Delta \leq t \leq \Delta, \\ 0 & \text{otherwise.} \end{cases} \tag{4.25}$$

for triangular dither. Then, since due to the rounding operations all values of s in the interval $[-\frac{\Delta}{2}, \frac{\Delta}{2})$ are quantized to zero, we can define the probability of an output value being zero based on the probability density of s as follows:

$$p(y = 0) = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} f_s(s)ds. \tag{4.26}$$

In order to evaluate the impact of the dither signal on the induced sparsity, we make some assumptions about the input pdf f_x. As we will see in the next chapter, the pre-activation gradients during the backward pass are distributed according to a zero-centered Laplace-like distribution. For our analysis we will therefore assume x follows a zero-centered Laplace distribution. We determine the theoretically induced sparsity by evaluating equation 4.26 numerically for different Laplace-distributed f_x and different quantization step sizes Δ. We also simulate the quantization of values drawn from the Laplace distribution and measured the resulting sparsity and error variance to confirm the theoretical results. Figure 4.3 shows exemplary results for a specific Laplace distribution, and theoretically induced sparsity for other Laplace distributions is shown in the appendix in figure C.1. We find that the use of uniform dither always

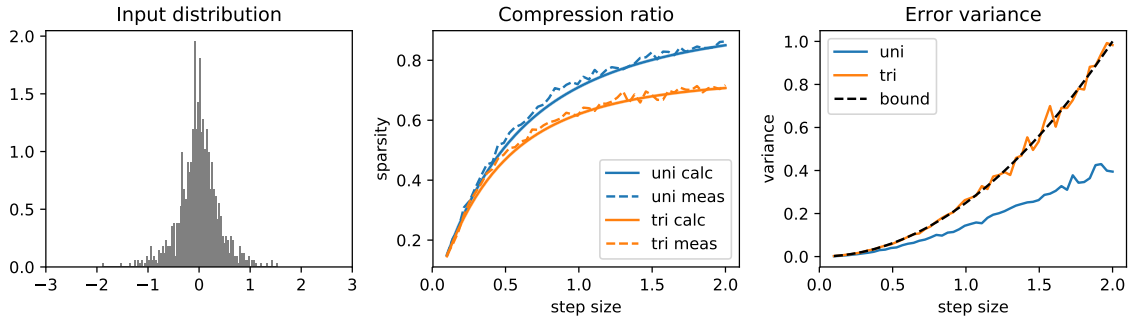

Figure 4.3: Sparsity (calculated and measured) and error variance of quantization with input values sampled from a Laplace distribution with mean $\mu = 0$ and scale $b = 0.3$ (histogram on the left), using uniform dither (uni) or triangular dither (tri).

induces higher sparsity for zero-centered Laplace distributions than using triangular dither. Especially for large quantization step sizes, the use of uniform dither can lead to high sparsity ratios. In the middle plot of figure 4.3 we see that the measured sparsity coincides with the the theoretical results. Furthermore, the right plot shows a significantly reduced error variance for uniform dither compared to triangular dither. Together with the fact that the measured error variance induced by triangular dither is given by $\Delta^2/4$ ("bound" in the plot), this confirms our theoretical discussion about the induced error variances from the last sections.

Concluding the comparison between uniform and triangular dither, we have shown that for relevant input distributions the use of uniform dither in a uniform quantization scheme induces higher sparsity ratios and significantly lower error variance as compared to triangular dither. As a result, despite its inferior ability to control the exact error variance, uniform dither could potentially be the better choice for our method. In the next chapter, we formally introduce Quantized Back Propagation and how it combines the different concepts that we have discussed so far to efficiently train neural networks.

5 Quantized Back Propagation

We introduce Quantized Back Propagation (QBP) which leverages quantization techniques from signal processing in order to more efficiently train neural networks. In this chapter we explain how our method unifies the concepts introduced so far to achieve this goal.

5.1 Method Overview

When training neural networks the back propagation algorithm is used to efficiently evaluate the gradients of the loss function for a given input by propagating local errors through the network. Since up to 90% of the computing time for training fully-connected neural networks is spent on the matrix multiplication operations [2], in this method we focus on reducing the computational cost of these operations. Recall from subsection 2.2.2 that for fully-connected neural networks, there are three matrix multiplications involved in the back propagation algorithm: One in the forward pass and two in the backward pass. Altering the computations in the forward pass would also change the activation values and the prediction error that is computed using the loss function, and thus have multiple effects on the backward pass as well. These are more difficult to analyze theoretically and thus for this method we focus on the two matrix multiplications in the backward pass. Both matrix multiplications in the backward pass involve the pre-activation gradients. In order to save operations, we apply a quantization function that compresses these gradients such that they exhibit a high degree of sparsity. We can then exploit this sparsity to omit operations when computing the matrix multiplications as described in section 2.3. The altered equations for the backward pass are given by

$$dz^l = da^l \odot f'(z^l), \tag{5.1}$$

$$dW^l = \widetilde{dz^l}(a^{l-1})^T, \tag{5.2}$$

$$da^{l-1} = (W^l)^T \widetilde{dz^l}. \tag{5.3}$$

with l being the current layer and $\widetilde{dz^l}$ the matrix of quantized pre-activation gradients. We use a uniform quantization function with non-subtractive uniform dither, as described in chapter 4, to element-wise quantize the values in the pre-activation gradient matrix:

$$\widetilde{dz^l_{ik}} = Q(dz^l_{ik} + \nu^l_{ik}) \tag{5.4}$$

$$= \Delta^l \lfloor \frac{dz^l_{ik} + \nu^l_{ik}}{\Delta^l} + \frac{1}{2} \rfloor \tag{5.5}$$

with Δ^l being the quantization step size, which can be chosen independently for in every layer, and ν^l_{ik} the dither signal sampled independently from a uniform distribution with a mean of zero and an amplitude equal to the quantization step size:

$$\nu^l_{ik} \sim \mathcal{U}(-\frac{\Delta^l}{2}, \frac{\Delta^l}{2}). \tag{5.6}$$

The motivation behind using this particular stochastic quantization function is that it induces a high sparsity on the pre-activation gradients, which results in high computational savings, and that the resulting weight gradients are unbiased and exhibit a bounded variance that can be controlled by the quantization step sizes. In the next sections we will go into more detail about these properties.

5.2 Uniform Quantization and Sparsity

In QBP we quantize the pre-activation gradients using a dithered uniform quantization scheme. However, uniform quantization is not necessarily a sparsifying quantization function, as compared to the top-k algorithm in [2] or Dropout [37]. Only input values that lie in the interval $[-\Delta/2, \Delta/2)$ are quantized to zero. Thus, uniform quantization can only achieve a high degree of output sparsity for input distributions that are centered around zero and using a sufficiently large quantization step size. It turns out that the pre-activation gradient values in ReLU-activated feed-forward networks are distributed according to a Laplace-like distribution with a mean of approximately zero. As we showed in section 4.3, for such input distributions, uniform quantization with uniformly distributed dither can achieve very high sparsity ratios using larger quantization step sizes. Figure 5.1 shows the Laplace-like distribution of the pre-activation gradients during training. More snapshots of pre-activation gradient distributions can be seen in appendix C.2.

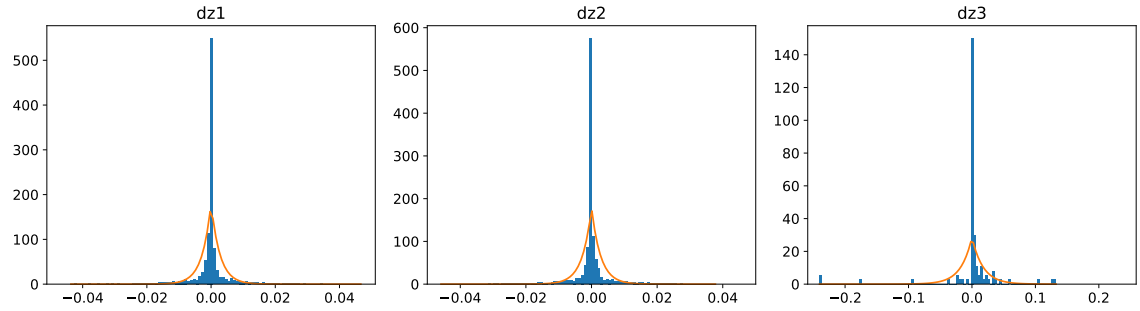

Figure 5.1: Pre-activation gradient distributions during training of a fully-connected ReLU-activated neural network with two hidden layers (500 neurons each) on MNIST.

5.3 Computational Cost

The per-layer complexity of QBP differs from regular back propagation in the following steps.

- Quantize dz^l element-wise - $\mathcal{O}(n_l m)$.
- Convert $\widetilde{dz^l}$ to the CSR memory format - $\mathcal{O}(n_l m)$.
- Compute dW^l leveraging sparse matrix multiplication - $\mathcal{O}(||\widetilde{dz^l}||_0 n_{l-1} + n_l n_{l-1})$.
- Compute da^{l-1} leveraging sparse matrix multiplication - $\mathcal{O}(||\widetilde{dz^l}||_0 n_{l-1} + n_{l-1} m + n_l)$.

We discussed the complexity of sparse matrix multiplications in section 2.3. Note that using the CSC memory format instead of CSR would be beneficial for dense-sparse multiplication but not for sparse-dense. However, the choice of memory format would not make a difference in the overall complexity of the algorithm, which is given by

$$E_{QBP} = \mathcal{O}(||\widetilde{dz^l}||_0 n_{l-1} + n_l n_{l-1} + n_l m + n_{l-1} m). \tag{5.7}$$

When we compare the time complexities of QBP and BP, which is $\mathcal{O}(n_l n_{l-1} m)$ as discussed in subsection 2.2.3, we can easily see that our method is asymptotically more efficient as long as $||\widetilde{dz^l}||_0 < n_l m$, which means as long as the quantized pre-activation gradients exhibit at least some sparsity. For high sparsity rates, we can achieve linear reduction in complexity.

5.4 Error Statistics

In this section we investigate whether QBP computes unbiased gradient estimates with bounded error variance, as desired. To understand how the quantization error induced on the pre-activation gradients is propagated through the network, we model it as additive noise:

$$\widetilde{dz^l} \equiv dz^l + \delta_p dz^l + \delta_q dz^l. \tag{5.8}$$

where $\delta_p dz^l$ denotes the propagated noise and $\delta_q dz^l$ denotes the noise added through quantization in the same layer. Then, the noise contained in the weight gradients, which is also the error of the method, is given by

$$\delta dW^l = \widetilde{dW^l} - dW^l \tag{5.9}$$

$$= \frac{1}{m}(\delta_p dz^l + \delta_q dz^l)(a^{l-1})^T. \tag{5.10}$$

Note that in the above formula we make the averaging of the weight gradients over the batch size, which is due to the use of SGD, explicit. We can now establish the following theorem regarding the statistical moments of the weight gradient noise:

Theorem. *Let $\delta_q dz^l$ be unbiased and uncorrelated quantization noise*

$$\mathbb{E}[\delta_q dz^l_{ik}] = 0 \quad \forall\ i,k,l \tag{5.11}$$

$$\mathbb{E}[\delta_q dz^l_{ik}\ \delta_q dz^{l'}_{i'k'}] = 0 \quad \forall\ i \neq i' \vee k \neq k' \vee l \neq l' \tag{5.12}$$

which is injected into the pre-activation gradients during back propagation. Then, the first and second moment of the induced gradient noise are given by

$$\mathbb{E}[\delta dW^l_{ij}] = 0 \quad \forall\ i,j,l \tag{5.13}$$

and

$$\mathbb{E}[(\delta dW^l_{ij})^2] = \frac{1}{m^2}\sum_{k=1}^{m}\Big(f'(a^l_{ik})^2\,\mathbb{E}[(\delta da^l_{ik})^2] + \mathbb{E}[(\delta_q dz^l_{ik})^2]\Big)\left(a^{l-1}_{jk}\right)^2, \tag{5.14}$$

with

$$\mathbb{E}[(\delta da^{l-1}_{jk})^2] = \sum_{i=1}^{n_l}\left(W^l_{ij}\right)^2\Big(f'(a^l_{ik})^2\,\mathbb{E}[(\delta da^l_{ik})^2] + \mathbb{E}[(\delta_q dz^l_{ik})^2]\Big). \tag{5.15}$$

We proof this theorem formally in appendix A. Consequently, if we apply an unbiased quantization scheme with uncorrelated quantization error to the pre-activation gradients during back propagation, we can guarantee that the resulting weight gradient estimates are unbiased and that the variance of the contained noise can be determined by the variance propagation formulas in equations 5.14 and 5.15. Then, it is trivial to show that using uniform quantization with uniform dither, as described in section 4.1, leads to unbiased weight gradient estimates that exhibit a bounded noise variance. We formalize this insight in the following corollary.

Corollary. *Given a quantization scheme as described in section 5.1, the first and second moment of the induced weight gradient noise are given by*

$$\mathbb{E}[\delta dW_{ij}^l] = 0 \quad \forall\, i, j, l \tag{5.16}$$

and

$$\mathbb{E}[(\delta dW_{ij}^l)^2] \leq \frac{1}{m^2} \sum_{k=1}^{m} f'(a_{ik}^l)^2\, \mathbb{E}[(\delta da_{ik}^l)^2](a_{jk}^{l-1})^2 + \frac{\Delta^2}{4m^2} ||a_{j*}^{l-1}||_2, \tag{5.17}$$

with

$$\mathbb{E}[(\delta da_{jk}^{l-1})^2] \leq \sum_{i=1}^{n_l} (W_{ij}^l)^2 f'(a_{ik}^l)^2\, \mathbb{E}[(\delta da_{ik}^l)^2] + \frac{\Delta^2}{4} ||W_{*j}^l||_2. \tag{5.18}$$

The variance bound in equations 5.17 and 5.18 follows from the fact that uniform quantization with uniform dither induces quantization noise with bounded variance, as described in the conjecture in subsection 4.2.3, and that all other terms in equations 5.14 and 5.15 are positive. Given the corollary from above, we have shown that our method fulfills the desired properties of unbiased noise with controllable variance. Note that the variance equation 5.14 for the weight gradient noise contains the factor $1/m^2$ and also a sum over m terms. As a result, the error variance induced by this method decreases linearly with the batch size. Note that when using triangular dither instead of uniform dither, the induced noise variance on the weight gradients is not bounded as we have shown in the corollary but it is exactly determined by the equations in the theorem.

5.5 Adaptive Quantization

As discussed in section 2.4, when training neural networks the ideal noise scale for the weight gradients in the different layers is not yet generally known. However, if insights about the theoretical noise scale or approximations of it become available through more research, QBP can be used to induce such a desired noise scale. Leveraging the theorem for bounded error variance that we introduced in the last section, we can design an algorithm that adaptively chooses the quantization step size in each layer to control the variance of the induced weight gradient noise.

Corollary. *Given a quantization scheme as described in section 5.1, let*

$$\mathbb{E}[(\delta dW_{ij}^l)^2] \leq \sigma^l \quad \forall\, i, j \tag{5.19}$$

be the desired variance bound for the induced weight gradient noise in layer l. Then, this bound holds when the quantization step size is chosen as

$$\Delta^l = \min_{i,j} \Gamma_{ij} \tag{5.20}$$

with $\Gamma \in \mathbb{R}^{n_l \times n_{l-1}}$ *and*

$$\Gamma_{ij} = \sqrt{\frac{4m^2\sigma^l - 4\sum_{k=1}^{m} f'(a_{ik}^l)^2\, \mathbb{E}[(\delta da_{ik}^l)^2](a_{jk}^{l-1})^2}{||a_{j*}^{l-1}||_2}}. \tag{5.21}$$

The definition of Γ_{ij} follows from rearranging equation 5.17 for Δ^l with $\mathbb{E}[(\delta dW_{ij}^l)^2] = \sigma^l$ and thus represents the required quantization step sizes such that noise variance σ^l is induced on any weight gradient in layer l. Since the induced noise variance is a monotonically increasing function of Δ^l, by choosing Δ^l as the minimum of the step sizes in Γ_{ij} we ensure that the maximum induced noise variance is equal to σ^l. This algorithm can be used to mimic noise that is artificially added to the weight gradients, such that QBP can be adapted to induce a desired noise scale on the weight gradients and thus compare to approaches like SGLD [33]. Note that because of the necessary variance propagation, the asymptotic complexity of such an algorithm is equivalent to regular back propagation. We leave the efficient approximations of the variance formulas for future work.

An alternative strategy for adaptive quantization would be to choose the quantization step size such that a desired sparsity constraint is fulfilled. For this, in every iteration, the distribution of pre-activation gradients is approximated by fitting a Laplace distribution to values. Then, the integrals from section 4.3 are solved to find the quantization step size that induces the desired number of zero-valued elements.

6 Experiments

In this chapter we present numerical experiments we conducted to investigate the properties of QBP. We train fully-connected neural networks on different image classification tasks and measure the error statistics and the computational benefit of our method as well as the generalization performance of the resulting networks.

6.1 Experimental Settings

The fully-connected networks in our experiments have two hidden layers of 500 units each. We train on three different benchmark datasets for image classification. The MNSIT digit recognition task [61] encompasses 60,000 training and 10,000 test images of handwritten digits, each gray-scale and 28 by 28 pixels. The corresponding labels assign the images to one of the ten digit classes. The CIFAR10 dataset [62] consists of 50,000 training and 10,000 test images of vehicles and animals that belong to ten different classes such as "automobile", "cat" and "horse". The images are 32 by 32 pixels and have RGB color channels. The SVHN dataset [63] encompasses around 73,000 training and around 26,000 test images of house number photographs that were cropped to contain single digits, and thus each belong to one of the ten digit categories. We use either SGD or Adam [64], a variant of SGD with adaptive learning rate that leverages exponential averaging of past gradients for the weight updates. For the hidden layers in the network we use a ReLU activation function, for the output layer we use the softmax function to predict class probabilities. We use the cross entropy loss function, which is common for classification tasks. To measure the generalization performance we calculate the prediction accuracy of the network based on a dedicated test dataset. This test accuracy is calculated using the model that attained the highest validation accuracy throughout the training procedure. The validation accuracy is measured on a dedicated set of 5000 training samples that were not included during training - the validation set. For MNIST, we train the networks for 25 passes through the training set (epochs) and use a batch size of 10, for CIFAR10 and SVHN we train for 50 epochs with a batch size of 50. We use a constant quantization step size for most of the experiments and scale the pre-activation gradients by the batch size before quantizing them.

6.2 Error Statistics

In order to verify the error statistics of our method we train a neural network with QBP on the MNIST dataset. We use SGD with a learning rate of $\eta = 10^{-3}$ and a constant quantization step size of $\Delta^l = 0.1$. Recall that the error of our method is given by the difference between the weight gradients computed by QBP and the weight gradients computed by regular back propagation (BP). In order to compute the error statistics, we sample from the distribution of weight gradient estimates at specific training iterations by performing additional QBP backward passes for the current training batch, which each gives us the gradients for every weight in the network. Then from these estimates

we subtract the true weight gradients computed by a regular BP backward pass for the same training batch. The resulting error samples represent the error distribution at a specific time in the training procedure. We usually use 50 additional QBP backward passes, which means we approximate the error distribution using 50 samples. We can then compute the mean and variance of these gradient error samples for the single weights in the network, which we call *error means* and *error variances*. To avoid inspecting the error of every weight gradient individually and also to be able to track the error statistics over the course of the training procedure, we compute "second-order" statistics. This means, that we compute the mean, the variance, and the maximum value of the error means and error variances over all weight gradients in a certain layer. Using these second-order statistics we can then infer if our method is unbiased and has bounded variance all weight gradients in individual layers. We compute these error statistics every 2000 training iterations. In the figures, we refer to these measures as, for example, "dW2 err var mean" for the mean of error variances for all weight gradients in layer two. We refer to the weights connecting the input layer to the first hidden layer as layer one (784x500 weights), to the weights connecting the two hidden layers as layer two (500x500 weights) and to the weights connecting the second hidden layer with the output layer as layer three (500x10 weights).

6.2.1 Mean

To show that QBP with uniform dither (QBP-UD) actually produces unbiased weight gradient estimates we compare it to QBP without dither signal (QBP-ND), which means that the pre-activation gradients are quantized using regular, deterministic uniform quantization, as described in section 4.1. The findings are illustrated in figure 6.1, where we exemplarily show the second-order statistics of the error means in the second layer. In the top row of the figure, we can see that the error mean is on average lower for QBP-UD than for QBP-ND (left plot) and that the error means of the single weight gradients deviate less from this average (right plot). The bottom row of the figure illustrates the same measurements for QBP-UD with a different number of samples for the computation of the error statistics. We can see here that the error means decrease for an increased sample count. We find very similar results for the other layers of the network, as can be seen in the appendix in figures C.3 and C.4. To summarize, the use of uniform dither in our method results in a significantly lower error mean, as compared to using no dither signal. Also the error mean is even lower for an increased number of error samples. Since the mean is an approximation of the expected value we conclude that the error of QBP-UD does indeed go to zero and the method is therefore unbiased.

6.2.2 Variance

Similarly to the error mean we also analyze the error variance of QBP. We compute the expected error variance induced by QBP with triangular dither (QBP-TD) and the variance bound for QBP-UD using the equations from section 5.4 and refer to it as the "calculated" error variances. We compare these calculated error variances to the actually induced ones for both methods and illustrate our findings in figure 6.2 We find that for uniform dither the maximum of the calculated error variance bound is always larger than the maximum of the induced error variance (top left plot). For triangular dither the calculated error variance is on average roughly equal to the induced one

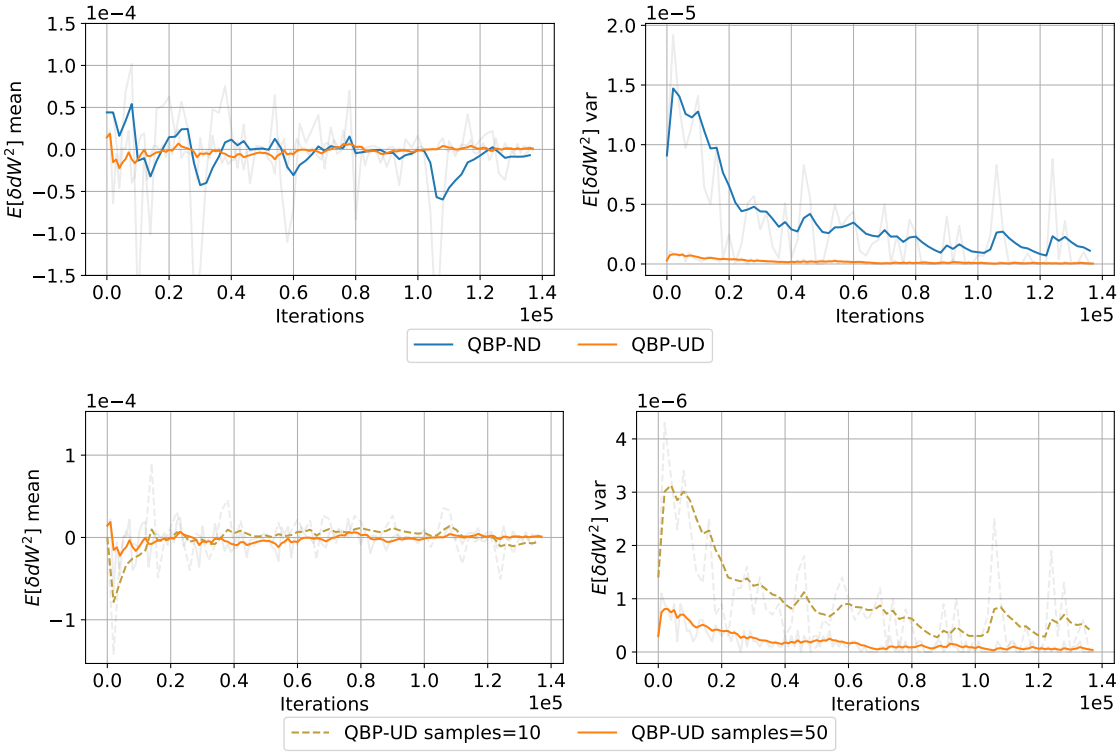

Figure 6.1: Fully-connected neural network with two hidden layers (500,500) trained on MNIST using QBP with constant quantization step size $\Delta^l = 0.1$. Plots show second-order statistics (mean and variance) over weights in second layer of weight gradient error means. Top row compares error mean using either no dither signal (QBP-ND) or non-subtractive uniform dither (QBP-UD). Bottom row compares error mean computed based on different sample sizes. The x-axis represents the training iterations. An exponential moving average with $\beta = 0.8$ was used for curve smoothing.

(top right plot). Furthermore, we find that doubling the batch size roughly halves the induced error variance for both methods (bottom plots).

From the analysis presented for mean and variance of the weight gradient error we can conclude that the gradient estimates computed by QBP fulfill the expected properties when the method is used to train neural networks on a benchmark dataset. We found that the gradient estimates are indeed unbiased, that the variance bound for the error induced by QBP with uniform dither holds and that the variance of the error induced by QBP with triangular dither matches the expected variance. This confirms our theoretical derivations in section 5.4.

6.3 Quality vs. Efficiency

So far we have analyzed the error statistics of QBP. However, more important for the practical application of this method is how well the trained networks generalize and how much computation can be saved by exploiting the sparsity induced by quantization. While strong quantization leads to high sparsity and thus to a large computational benefit, it also adds a lot of noise to the weight gradients which can lead to a destabilization of the training process and therefore to networks that generalize badly. To

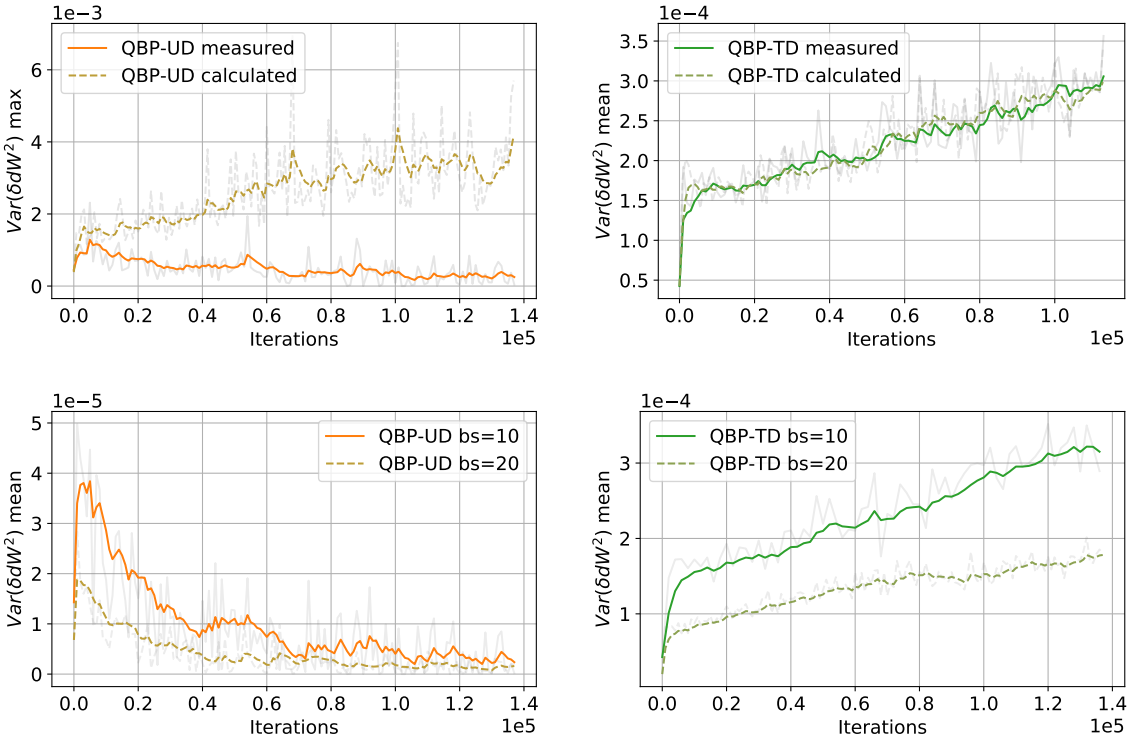

Figure 6.2: Fully-connected neural network with two hidden layers (500,500) trained on MNIST using QBP with constant quantization step size $\Delta^l = 0.1$ and either uniform (QBP-UD) or triangular dither (QBP-TD). Plots show second-order statistics (mean and max) over weights in second layer of either calculated or measured weight gradient error variances. Calculated error variances are obtained from variance formulas in section 5.4. The x-axis represents the training iterations. An exponential moving average with $\beta = 0.8$ was used for curve smoothing.

investigate this trade-off we test different choices for the quantization step size. Since the sparsity ratio in the pre-activation gradients directly determines the computational benefit of QBP, as discussed in section 5.3, we measure the averaged sparsity over layers and training iterations in order to approximate the efficiency gain of the method.

6.3.1 Comparison of Dither Signals

In order to analyze the impact of different dither signals we train a neural network on the MNIST dataset with QBP and either no dither signal, uniform dither or triangular dither. We use SGD with a learning rate of $\eta = 10^{-3}$ and different constant quantization step sizes $\Delta^l = s$. We illustrate our findings in figure 6.3.

Undithered Quantization

We found that using no dither signal works well for smaller quantization step sizes and results in generalization-sparsity ratios roughly comparable to uniform dither. However, for larger step sizes, we find that networks trained without dither signal do not converge to a good solution and thus exhibit drastically reduced generalization performance. We can see from the left plot that, for this experiment, the performance degradation of

undithered quantization starts for quantization step sizes around $s = 1$. For uniform dither, this drastic performance degradation does not occur even for much larger step sizes. We conjecture that the performance degradation for undithered quantization is due to the biased quantization error which is less pronounced for smaller quantization step sizes. This is in line with the classical model of quantization, which we described in section 4.1, and which states that for small enough quantization step sizes relative to the input the quantization error is unbiased even without the use of a dither signal. We thus compared the error mean induced by using no dither for the quantization step sizes where the performance degradation begins and found, as illustrated in figure 6.4, that for $s = 1$ the induced error mean is much larger than for $s = 0.33$, with the later still being comparable to the error mean induced by uniform dither. We can conclude that to achieve high computational savings, the use of dither is necessary to keep the quantization error unbiased and thus the training procedure stable.

Triangular Dither

We also found that the use of triangular dither, when compared to uniform dither, leads to a decrease in generalization performance and a drastic decrease in induced sparsity for the same quantization step sizes. The drastic decrease in sparsity confirms our simulations for Laplace-like input distributions from section 4.3. Furthermore, we explain the weaker generalization with the significantly higher amount of noise variance triangular dither injects into the weight gradients computed by QBP, which we also showed experimentally in subsection 6.2.2. The noise modulation that results from using uniform dither, which we discussed in subsection 4.2.3, seems to either not be too disadvantageous for the training procedure or is actually beneficial. This noise modulation could represent a helpful pre-conditioning of the gradient noise that is caused by the back propagation algorithm itself. In fact, as we can see in figure 6.2, the error variance induced by using triangular dither is increasing throughout the training procedure whereas the error variance induced by using uniform is decreasing. Potentially, this is due to the decreasing amplitude of the pre-activation gradients during training, which then affects the input-dependent error variance induced by using uniform dither, but not the input-independent variance induced by using triangular dither. Such an annealing of the noise contained in the weight gradients is also recommended by approaches such as SGLD [33] which add controlled artificial noise to the weight gradients during training to converge to better solutions with higher probability. Taken together, these findings show that uniform dither represents a superior choice of dither signal for QBP as compared to triangular dither. Also, these findings suggest that QBP with uniform dither does indeed add noise to the weight gradients that is similar to the noise added by SGD - it is unbiased and exhibits a pre-conditioning of the variance that is beneficial for the training procedure.

6.3.2 Comparison with meProp

In order to analyze the quality-efficiency trade-off we compare our method with regular BP and with meProp [2]. We use Adam with a base learning rate of $\eta_0 = 10^{-4}$ for BP and QBP, and $\eta_0 = 10^{-3}$ for meProp, as it is also reported in their paper. The other hyperparameters for Adam are kept to their default values. We evaluate the different methods on the MNIST, CIFAR and SVHN datasets. Figure 6.5 illustrates the quality-

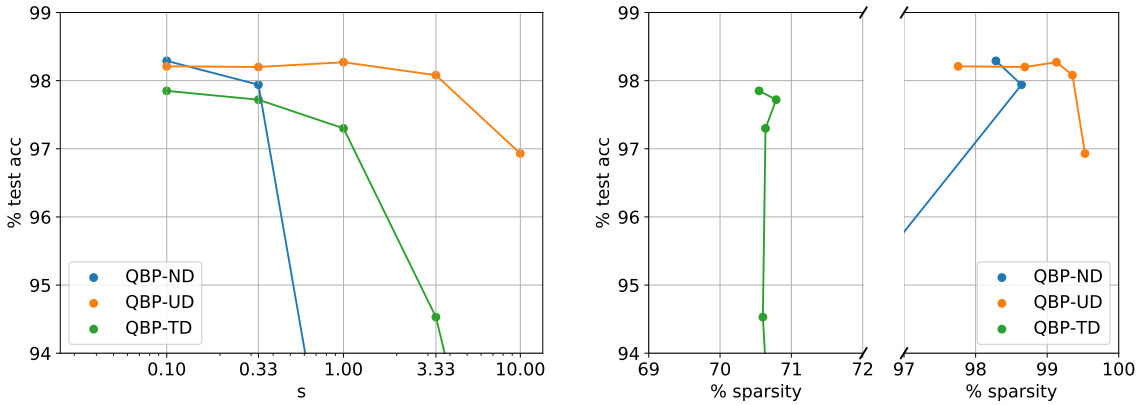

Figure 6.3: Fully-connected neural network with two hidden layers (500,500) trained on MNIST using QBP with constant quantization step sizes $\Delta^l = s$ and either no dither signal (QBP-ND), uniform dither (QBP-UD) or triangular dither (QBP-TD). Sparsity is averaged over layers and training iterations. Training results represented by points in the left plot correspond to training results in the right plot.

efficiency ratios on the different datasets. We find that on MNIST, QBP can induce an average sparsity in the pre-activation gradients of around 99% while still obtaining the same accuracy as regular back propagation, and around 77% and 92% on CIFAR10 and SVHN respectively. meProp however cannot recover the full accuracy of BP for any of the tasks. We summarized results obtained on the MNIST dataset in table 6.1.

Algorithm	Hyperparameter	Test accuracy (%)	Average sparsity (%)
BP	-	98.13 ± 0.133	-
QBP-UD	$\Delta^l = 1$	$\mathbf{98.14 \pm 0.156}$	$\mathbf{99.15 \pm 0.029}$
meProp	$k = 50$	97.89 ± 0.171	94.11 ± 0.033

Table 6.1: Results from training a fully-connected neural network with two hidden layers (500,500) on the MNIST dataset using different variants of the back propagation algorithm. Sparsity contained in the quantized pre-activation gradients is averaged over layers and training iterations.

6.4 Adaptive Quantization

To investigate the ability of our method to control the noise induced on the parameter gradients, we train a fully-connected network on MNIST as described in section 6.1 with QBP and uniform dither, and adaptively choose the quantization step size according to the algorithm from section 5.5 such that a desired variance bound of $\sigma^l_{max} = 0.01$ is met. Figure 6.6 illustrates the resulting error statistics of the weight gradients in the first layer at iteration 2000 during the training procedure. As we can see, most weight gradients have an error mean of zero and an error variance below the desired bound. Less than 0.4% of the weight gradients have error variance higher than σ^l_{max}. Inspecting the error distributions at different points of time during training yields similar results.

From these experiments, we can conclude that QBP can be applied to induce a

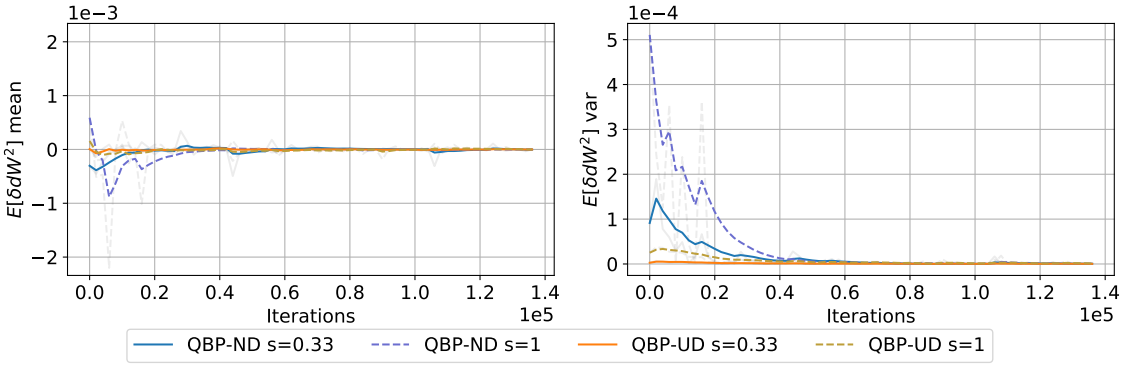

Figure 6.4: Fully-connected neural network with two hidden layers (500,500) trained on MNIST using QBP with two choices for the constant quantization step size $\Delta^l = s$ and either no dither signal (QBP-ND) or uniform dither (QBP-UD). Plots show second-order statistics (mean and max) over weights in second layer of weight gradient error means. The x-axis represents the training iterations. An exponential moving average with $\beta = 0.8$ was used for curve smoothing.

desired noise scale in the weight gradients. Therefore, given knowledge about an optimal noise scale for stochastically training neural networks, our method uses adaptive quantization to achieve the best possible trade-off between efficiency and generalization performance of the resulting models.

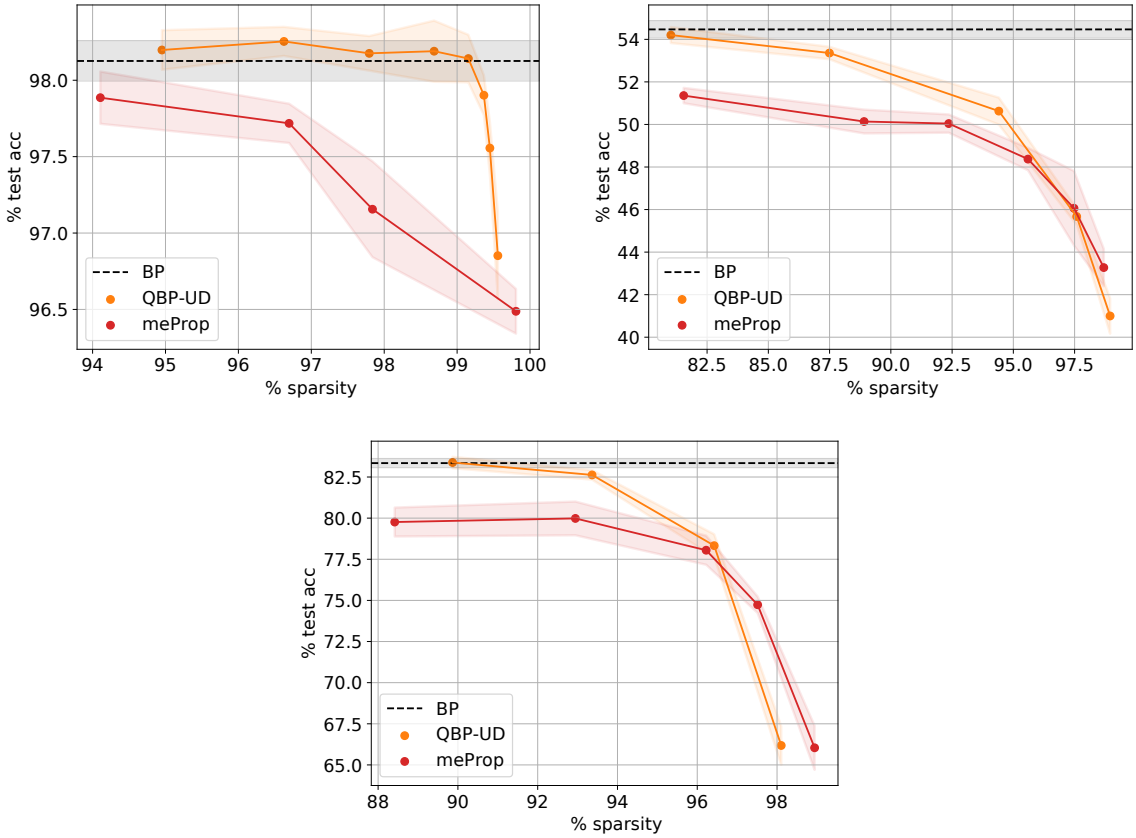

Figure 6.5: Fully-connected neural network with two hidden layers (500,500) trained on MNIST, CIFAR10 and SVHN using either regular back propagation (BP), QBP with uniform dither (QBP-UD) or meProp [2]. For QBP different choices for a constant quantization step size $\Delta^l = s$ were used, and for meProp different choices for the k hyperparameter, which controls the induced sparsity. Sparsity is averaged over layers and training iterations. Multiple runs with different random seeds were executed for each configuration. Points show mean performance with standard deviation indicated as span.

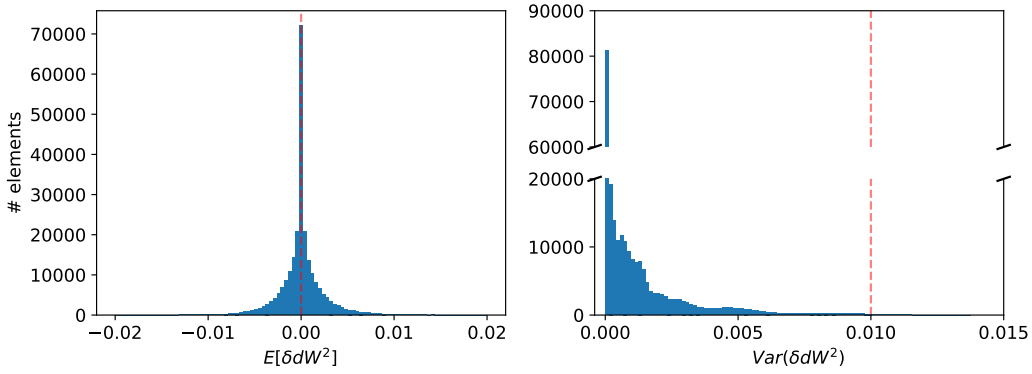

Figure 6.6: Fully-connected neural network with two hidden layers (500,500) trained on MNIST using QBP with variance-constraint adaptive quantization and a desired variance bound of $\sigma^l = 0.01$. Histograms show the means (left) and variances (right) of the gradient errors in the second layer.

7 Conclusion

The research question for this thesis was how to use quantization to efficiently train neural networks without loss in generalization performance or with as little as possible. Our approach to this problem was to leverage a stochastic quantization scheme during the back propagation algorithm that induces high sparsity and injects noise into the weight gradients with similar characteristics as noise induced by SGD. We motivate these noise characteristics with the conjecture that we discuss in section 2.4, which states that SGD works so well for training neural networks because the resulting weight gradient estimates are unbiased and exhibit noise variance that is well pre-conditioned to the optimization landscape. To implement this approach we make use of dithered uniform quantization and exploit the resulting sparsity in the quantized pre-activation gradients by computing efficient sparse matrix multiplications. We prove that the resulting weight gradient estimates are unbiased and exhibit controllable noise variance, as stated more generally by our theorem in section 5.4. In our experiments we verify that these noise statistics are aligned with our theoretical predictions when we apply our method in practice. We demonstrate that, as conjectured previously, unbiased gradient estimates are important for good generalization-efficiency trade-offs by outperforming deterministic approaches, which we illustrate in figures 6.3 and 6.5. We also show that, similarly to SGD, our method provides a useful noise pre-conditioning by comparing dither signals experimentally in subsection 6.3.1. Furthermore, our method can exhibit noise scales with arbitrary bounds by adaptive quantization, as demonstrated in section 6.4, and therefore in principle presents an optimal compromise between generalization and efficiency. In conclusion, we established a novel framework for stochastic quantization with controlled statistical properties during neural network training and leveraged this framework to achieve state-of-the-art results on multiple benchmark datasets.

8 Future Work

We introduced a method that leverages stochastic quantization to reduce the computational cost of training neural networks while exhibiting desired error statistics. However, there are still many unanswered questions regarding the practical applications of this method and its generalization to different network architectures as well as potential theoretical improvements. More research is necessary to find the optimal noise scale for stochastically training neural networks. Given efficient approximations of the propagated variances, our method could then be used to meet this noise scale while maximally increasing the training efficiency. Furthermore, stochastic quantization of the pre-activation gradients can be compared to regularization techniques such as dropout and data augmentation, which also aim to achieve better generalizing network models. It would be interesting to see if gradient noise injected by our method exhibits similar properties to these successful and commonly used heuristics. A potential advancement of our method is to also apply the quantization scheme to the other matrices involved in the backward pass - the weight matrix and the input gradients - to leverage even more efficient sparse-sparse matrix multiplications. Another possibility is the additional quantization of the weight matrix and the activation matrix in the forward pass in order to reduce the computational cost in both forward and backward pass. However, this would complicate the calculation of the induced error variance because the loss signal would change and so would the propagated gradients. More future work should also be directed at applying stochastic quantization to more types of network architectures. For example, this work can be easily applied to recurrent neural networks (RNNs) since they usually consist of fully-connected layers as well and thus define a similar backward pass as discussed here. Since convolution operations can be modeled as matrix multiplications [65], our method transfers easily to convolutional neural networks (CNN). Moreover, this work can be applied to settings of distributed training where the noise contained in the weight gradients can be leveraged for more efficient communication between clients.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[2] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. *meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting*. 2017. arXiv: 1706.06197.

[3] Donald O. Hebb. "The Organization of Behavior: A Neuropsychological Theory". In: *A Wiley Book in Clinical Psychology*. (1949), pp. 62–78.

[4] Frank Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* 65.6 (1958), p. 386.

[5] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 2006.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer Feedforward Networks are Universal Approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[8] Andrej Karpathy and Li Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3128–3137.

[9] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML)*. Omnipress, 2010, pp. 807–814.

[10] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[11] Grégoire Montavon, Genevive Orr, and Klaus-Robert Müller. *Neural Networks: Tricks of the Trade*. Springer Publishing Company, Inc., 2012.

[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*. Vol. 1. MIT press Cambridge, 2016.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), p. 533.

[14] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. "Basic linear algebra subprograms for Fortran usage". In: *ACM Transactions on Mathematical Software (TOMS)* 5.3 (1979), pp. 308–323.

[15] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in pytorch". In: *NIPS 2017 Workshop Autodiff talk* (2017).

[17] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: a system for large-scale machine learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[18] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2017. arXiv: 1706.02677.

[19] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001. URL: http://www.scipy.org.

[20] Richard Dorrance, Fengbo Ren, and Dejan Marković. "A scalable sparse matrix-vector multiplication kernel for energy-efficient sparse-blas on FPGAs". In: *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*. ACM. 2014, pp. 161–170.

[21] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2016. arXiv: 1609.04836.

[22] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. *Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition*. 2015. arXiv: 1503.02101.

[23] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. *How to Escape Saddle Points Efficiently*. 2017. arXiv: 1703.00887.

[24] Moritz Hardt, Benjamin Recht, and Yoram Singer. "Train Faster, Generalize Better: Stability of Stochastic Gradient Descent". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML)*. Vol. 48. 2016, pp. 1225–1234.

[25] Wenlong Mou, Liwei Wang, Xiyu Zhai, and Kai Zheng. "Generalization Bounds of SGLD for Non-convex Learning: Two Theoretical Viewpoints". In: *Proceedings of the 31st Conference On Learning Theory*. Vol. 75. Proceedings of Machine Learning Research. PMLR, 2018, pp. 605–638.

[26] Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Karthik Sridharan, Brando Miranda, Noah Golowich, and Tomaso Poggio. *Theory of Deep Learning III: Generalization Properties of SGD*. Tech. rep. Center for Brains, Minds and Machines (CBMM), 2017.

[27] Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. "An Alternative View: When Does SGD Escape Local Minima?" In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 2698–2707.

[28] Sepp Hochreiter and Jürgen Schmidhuber. "Flat minima". In: *Neural Computation* 9.1 (1997), pp. 1–42.

[29] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. *Entropy-SGD: Biasing Gradient Descent Into Wide Valleys*. 2016. arXiv: 1611.01838.

[30] Alessandro Achille and Stefano Soatto. *Emergence of Invariance and Disentanglement in Deep Representations*. 2017. arXiv: 1706.01350.

[31] Samuel L. Smith and Quoc V. Le. *A Bayesian Perspective on Generalization and Stochastic Gradient Descent*. 2017. arXiv: 1710.06451.

[32] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. *Snapshot Ensembles: Train 1, get M for free*. 2017. arXiv: 1704.00109.

[33] Max Welling and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML)*. Omnipress, 2011, pp. 681–688.

[34] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. *Adding Gradient Noise Improves Learning for Very Deep Networks*. 2015. arXiv: 1511.06807.

[35] Chunyuan Li, Changyou Chen, David E Carlson, and Lawrence Carin. "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks." In: *AAAI*. Vol. 2. 3. 2016, p. 4.

[36] Gaétan Marceau-Caron and Yann Ollivier. *Natural Langevin Dynamics for Neural Networks*. 2017. arXiv: 1712.01076.

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[38] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. *Training deep neural networks with low precision multiplications*. 2014. arXiv: 1412.7024.

[39] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep Learning with Limited Numerical Precision". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning (ICML)*. Vol. 37. 2015, pp. 1737–1746.

[40] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015, pp. 3123–3131.

[41] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *European Conference on Computer Vision (ECCV)*. 2016, pp. 525–542.

[42] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*. 2016. arXiv: 1602.02830.

[43] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R. Iris Bahar, and Sherief Reda. "Understanding the Impact of Precision Quantization on the Accuracy and Energy of Neural Networks". In: *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1478–1483.

[44] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. *DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients*. 2016. arXiv: 1606.06160.

[45] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations". In: *Journal on Machine Learning Research* 18.1 (2017), pp. 6869–6898.

[46] X. Chen, X. Hu, H. Zhou, and N. Xu. "FxpNet: Training a deep convolutional neural network in fixed-point representation". In: *International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2494–2501.

[47] Urs Köster et al. "Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks". In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 1742–1752.

[48] Paulius Micikevicius et al. "Mixed Precision Training". In: *International Conference on Learning Representations (ICML)*. 2018.

[49] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[50] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. "Training and Inference with Integers in Deep Neural Networks". In: *International Conference on Learning Representations (ICLR)*. 2018.

[51] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R. Aberger, Kunle Olukotun, and Christopher Ré. *High-Accuracy Low-Precision Training*. 2018. arXiv: 1803.03383.

[52] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. *QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding*. 2016. arXiv: 1610.02132.

[53] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning". In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 1509–1519.

[54] Menachem Adelman and Mark Silberstein. *Faster Neural Network Training with Approximate Tensor Operations*. 2018. arXiv: 1805.08079.

[55] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. "Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication". In: *SIAM Journal on Computing* 36.1 (2006), pp. 132–157.

[56] Leonard Schuchman. "Dither signals and their effect on quantization noise". In: *IEEE Transactions on Communication Technology* 12.4 (1964), pp. 162–165.

[57] Stanley P. Lipshitz, Robert A. Wannamaker, and John Vanderkooy. "Quantization and dither: A theoretical survey". In: *Journal of the Audio Engineering Society* 40.5 (1992), pp. 355–375.

[58] Robert Alexander Wannamaker. "The Theory of Dithered Quantization". PhD thesis. 1997.

[59] Robert A. Wannamaker, Stanley P. Lipshitz, John Vanderkooy, and J. Nelson Wright. "A theory of nonsubtractive dither". In: *IEEE Transactions on Signal Processing* 48.2 (2000), pp. 499–516.

[60] N. S. Jayant and Peter Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video, Prentice-Hall Signal Processing Series*. 1984.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[62] Alex Krizhevsky and Geoffrey Hinton. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. Citeseer, 2009.

[63] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning". In: *NIPS workshop on deep learning and unsupervised feature learning*. 2. 2011, p. 5.

[64] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. 2015.

[65] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

A Error Statistics of Quantized Back Propagation

In this chapter we proof the theorem given in section 5.4.

A.1 Error Propagation

We model the additive quantization noise contained in the pre-activation gradients during the backward pass as

$$\widetilde{dz^l} = dz^l + \delta_p dz^l + \delta_q dz^l \tag{A.1}$$

with $\delta_q dz^l$ being the noise induced by quantization in the current layer and $\delta_p dz^l$ the noise induced by quantization in upper layers. Then, the noise contained in the weight gradients is given by

$$\begin{aligned} \delta dW^l &= \widetilde{dW^l} - dW^l && \text{(A.2)} \\ &= \frac{1}{m}(\delta_p dz^l + \delta_q dz^l)(a^{l-1})^T. && \text{(A.3)} \end{aligned}$$

Analogously, the noise contained in the propagated activation gradients is given by

$$\begin{aligned} \delta da^{l-1} &= \widetilde{da^{l-1}} - da^{l-1} && \text{(A.4)} \\ &= (W^l)^T(\delta_p dz^l + \delta_q dz^l). && \text{(A.5)} \end{aligned}$$

and thus the propagated noise in the pre-activation gradients is

$$\begin{aligned} \delta_p dz^l &= f'(a^l) \odot \widetilde{da^l} - dz^l \\ &= f'(a^l) \odot \delta da^l. && \text{(A.6)} \end{aligned}$$

Note that there is no noise propagated from the loss function

$$\delta da^L = \mathbf{0} \tag{A.7}$$

and thus

$$\mathbb{E}[\delta da^L_{ik}] = \mathbb{E}[(\delta da^L_{ik})^2] = 0 \quad \forall\ i, k. \tag{A.8}$$

A.2 Error Mean

Let the first moment and the joint first moment of the quantization error be given by

$$\mathbb{E}[\delta_q dz^l_{ik}] = 0 \quad \forall\ i, k, l, \tag{A.9}$$

$$\mathbb{E}[\delta_q dz^l_{ik}\ \delta_q dz^{l'}_{i'k'}] = 0 \quad \forall\ i \neq i' \vee k \neq k' \vee l \neq l'. \tag{A.10}$$

Then, using equations A.3, A.6 and A.9, it is straight-forward to describe the mean of the parameter gradient error as follows:

$$\begin{aligned}\mathbb{E}[\delta dW_{ij}^l] &= \frac{1}{m}\sum_k (\mathbb{E}[\delta_p dz_{ik}^l] + \mathbb{E}[\delta_q dz_{ik}^l]) a_{jk}^{l-1} \\ &= \frac{1}{m}\sum_k f'(a_{ik}^l)\mathbb{E}[\delta da_{ik}^l] a_{jk}^{l-1} \end{aligned} \tag{A.11}$$

As we can see from equation A.11, δdW^l depends on δda^{l-1} which we spell out similarly using equations A.5, A.6 and A.9:

$$\begin{aligned}\mathbb{E}[\delta da_{jk}^{l-1}] &= \sum_i W_{ij}^l (\mathbb{E}[\delta_p dz_{ik}^l] + \mathbb{E}[\delta_q dz_{ik}^l]) \\ &= \sum_k W_{ij}^l f'(a_{ik}^l)\mathbb{E}[\delta da_{ik}^l] \end{aligned} \tag{A.12}$$

From equation A.8, we can follow

$$\mathbb{E}[\delta da_{ik}^l] = 0 \quad \forall\, i,k,l, \tag{A.13}$$

and thus also

$$\mathbb{E}[\delta dW_{ij}^l] = 0 \quad \forall\, i,j,l. \tag{A.14}$$

A.3 Error Variance

Assume again that the first and joint first moment of the quantization error is zero, as stated in the last section. Then, to derive the second moment for the weight gradient error we first derive the second moment for the propagated activation gradient. We expand the second moment of δda^{l-1} using equations A.5 and A.6.

$$\begin{aligned}\mathbb{E}[(\delta da_{jk}^{l-1})^2] &= \mathbb{E}\Big[\Big(\sum_i W_{ij}^l (f'(a_{ik}^l)\delta da_{ik}^l + \delta_q dz_{ik}^l)\Big)^2\Big] \\ &= \sum_{i,i'} W_{ij}^l W_{i'j}^l \Big(f'(a_{ik}^l) f'(a_{i'k}^l)\mathbb{E}[\delta da_{ik}^l\, \delta da_{i'k}^l] + f'(a_{ik}^l)\mathbb{E}[\delta da_{ik}^l\, \delta_q dz_{i'k}^l] \\ &\quad + f'(a_{i'k}^l)\mathbb{E}[\delta_q dz_{ik}^l\, \delta da_{i'k}^l] + \mathbb{E}[\delta_q dz_{ik}^l\, \delta_q dz_{i'k}^l]\Big) \end{aligned} \tag{A.15}$$

To further simplify this equation, in the following we investigate the joint means of the noise terms. The joint mean between the propagated noise and the quantization noise can be written as

$$\begin{aligned}\mathbb{E}[\delta da_{ik}^l\, \delta_q dz_{i'k}^l] &= \mathbb{E}\Big[\sum_h W_{hi}^{l+1}\Big(f'(a_{hk}^{l+1})\delta da_{hk}^{l+1} + \delta_q dz_{hk}^{l+1}\Big)\delta_q dz_{i'k}^l\Big] \\ &= \sum_h W_{hi}^{l+1}\Big(f'(a_{hk}^{l+1})\mathbb{E}[\delta da_{hk}^{l+1}\, \delta_q dz_{i'k}^l] + \mathbb{E}[\delta_q dz_{hk}^{l+1}\, \delta_q dz_{i'k}^l]\Big) \\ &= \sum_i W_{hi}^{l+1} f'(a_{hk}^{l+1})\mathbb{E}[\delta da_{hk}^{l+1}\, \delta_q dz_{i'k}^l] \\ &= 0 \end{aligned} \tag{A.16}$$

using equations A.5, A.6 and A.10, and where in the last line we used a similar reasoning as in the last section, namely that the equation unrolls until it includes the last layer $l = L$ for which the propagated noise is zero and thus

$$\mathbb{E}[\delta da_{hk}^{L}\, \delta_q dz_{jk}^{l}] = 0 \quad \forall\ h, j, k, l. \tag{A.17}$$

Now, using equations A.10 and A.16 we can simplify equation A.15.

$$\begin{aligned}
\mathbb{E}[(\delta da_{jk}^{l-1})^2] &= \sum_{i,i'} W_{ij}^{l} W_{i'j}^{l} \Big(f'(a_{ik}^{l}) f'(a_{i'k}^{l}) \mathbb{E}[\delta da_{ik}^{l}\, \delta da_{i'k}^{l}] + \mathbb{E}[\delta_q dz_{ik}^{l}\, \delta_q dz_{i'k}^{l}] \Big) \\
&= \sum_{i,i'} \delta_{i \neq i'} W_{ij}^{l} W_{i'j}^{l} f'(a_{ik}^{l}) f'(a_{i'k}^{l}) \mathbb{E}[\delta da_{ik}^{l}\, \delta da_{i'k}^{l}] \\
&\quad + \sum_{i} (W_{ij}^{l})^2 \Big(\big(f'(a_{ik}^{l}) \big)^2 \mathbb{E}[(\delta da_{ik}^{l})^2] + \mathbb{E}[(\delta_q dz_{ik}^{l})^2] \Big) \\
&= \sum_{i} \big(W_{ij}^{l} \big)^2 \Big(\big(f'(a_{ik}^{l}) \big)^2 \mathbb{E}[(\delta da_{ik}^{l})^2] + \mathbb{E}[(\delta_q dz_{ik}^{l})^2] \Big)
\end{aligned} \tag{A.18}$$

For the last line we unrolled the gradients again to the last layer such that the term with $\delta_{i \neq i'}$ vanishes. Finally, using equations A.3, A.10 and A.16 we can derive the second moment of the weight gradient noise very similarly to $\mathbb{E}[(\delta dW_{ij}^{l})^2]$.

$$\begin{aligned}
\mathbb{E}[(\delta dW_{ij}^{l})^2] &= \frac{1}{m^2} \mathbb{E}\Big[\Big(\sum_{k} \big(f'(a_{ik}^{l}) \delta da_{ik}^{l} + \delta_q dz_{ik}^{l} \big) a_{jk}^{l-1} \Big)^2 \Big] \\
&= \frac{1}{m^2} \sum_{k,k'} \Big(f'(a_{ik}^{l}) f'(a_{ik'}^{l}) \mathbb{E}[\delta da_{ik}^{l}\, \delta da_{ik'}^{l}] + f'(a_{ik}^{l}) \mathbb{E}[\delta da_{ik}^{l}\, \delta_q dz_{ik'}^{l}] \\
&\quad + f'(a_{ik'}^{l}) \mathbb{E}[\delta_q dz_{ik}^{l}\, \delta da_{ik'}^{l}] + \mathbb{E}[\delta_q dz_{ik}^{l}\, \delta_q dz_{ik'}^{l}] \Big) a_{jk}^{l-1} a_{jk'}^{l-1} \\
&= \frac{1}{m^2} \sum_{k,k'} \Big(f'(a_{ik}^{l}) f'(a_{ik'}^{l}) \mathbb{E}[\delta da_{ik}^{l}\, \delta da_{ik'}^{l}] + \mathbb{E}[\delta_q dz_{ik}^{l}\, \delta_q dz_{ik'}^{l}] \Big) a_{jk}^{l-1} a_{jk'}^{l-1} \\
&= \frac{1}{m^2} \sum_{k,k'} \delta_{k \neq k} f'(a_{ik}^{l}) f'(a_{ik'}^{l}) \mathbb{E}[\delta da_{ik}^{l}\, \delta da_{ik'}^{l}] a_{jk}^{l-1} a_{jk'}^{l-1} \\
&\quad + \frac{1}{m^2} \sum_{k} \Big(\big(f'(a_{ik}^{l}) \big)^2 \mathbb{E}[(\delta da_{ik}^{l})^2] + \mathbb{E}[(\delta_q dz_{ik}^{l})^2] \Big) \big(a_{jk}^{l-1} \big)^2 \\
&= \frac{1}{m^2} \sum_{k} \Big(\big(f'(a_{ik}^{l}) \big)^2 \mathbb{E}[(\delta da_{ik}^{l})^2] + \mathbb{E}[(\delta_q dz_{ik}^{l})^2] \Big) \big(a_{jk}^{l-1} \big)^2
\end{aligned} \tag{A.19}$$

Equations A.14, A.18 and A.19 resemble the statements made in the theorem.

B Additional Experiments

B.1 Uniform Dither Variance Bound

In order to investigate the error variance induced by dithered uniform quantization with uniform and triangular dither, we simulate the quantization of values that are sampled from different probability distributions. In every run we sample a matrix of 10,000 values from the respective distribution, quantize them using a given quantization step size and compute the quantization error for every element. We do this for 1,000 runs and compute the error variance for every element in the matrix. In the bottom row of figure B.1 we illustrated the histogram of error variances over the elements in the matrix for different input distribution. We can see that for every distribution, the error variances induces by the use of uniform dither are bounded by $\Delta^2/4$, which is also the mean error variance when using triangular dither. Here, we used the quantization step size $\Delta = 1$. However, we find very similar results for other choices of quantization step size. This confirms our conjecture about the bounded error variance induced by uniform quantization with uniform dither, as described in subsection 4.2.3.

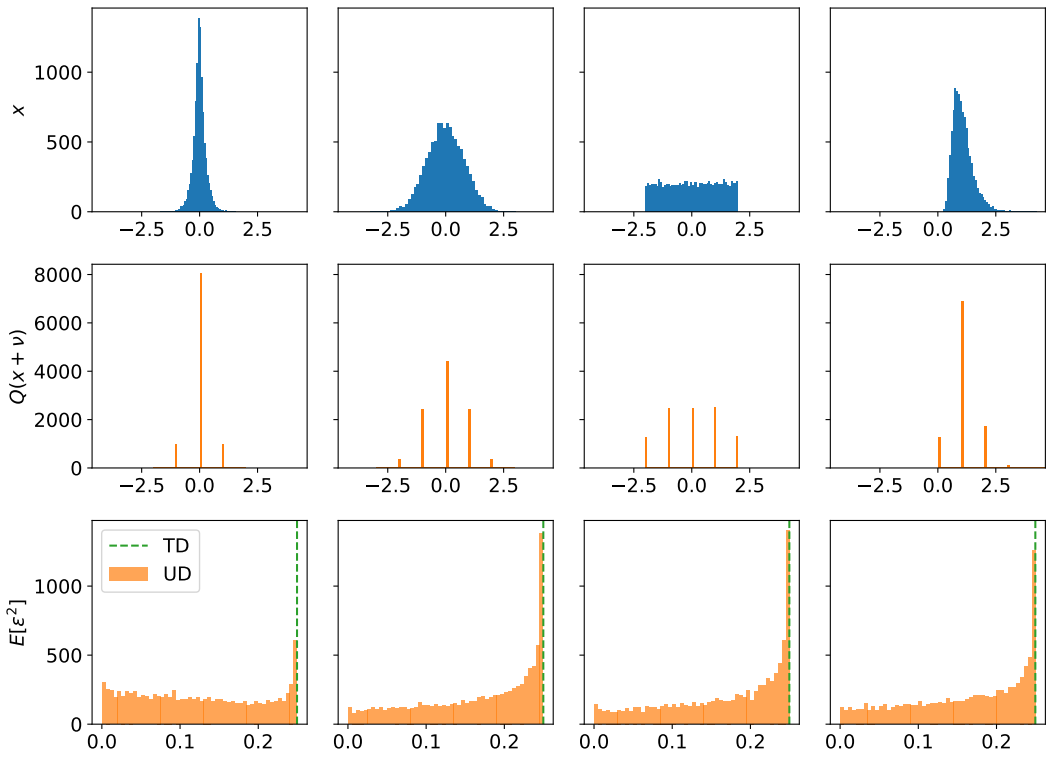

Figure B.1: Simulations of applying dithered uniform quantization with uniform dither signal to different input distributions. Top row shows input distribution input samples are drawn from in each iteration. Middle row shows quantization output distribution exemplarily for first iteration. Bottom row shows quantization error variances over iterations as histograms.

C Additional Figures

C.1 Uniform Quantization Sparsity

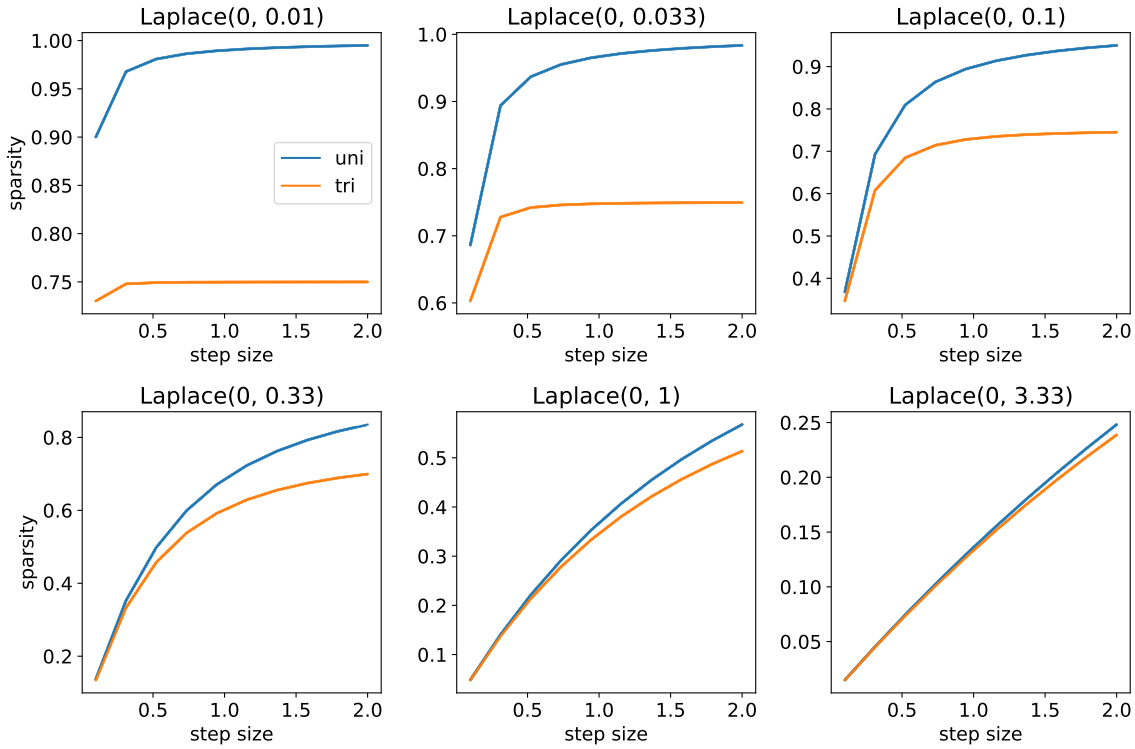

Figure C.1: Theoretical sparsity induced by uniform quantization with either uniform dither (uni) or triangular dither (tri), for different quantization step sizes, on input samples drawn from different zero-centered Laplace distributions.

C.2 Pre-Activation Gradient Distributions

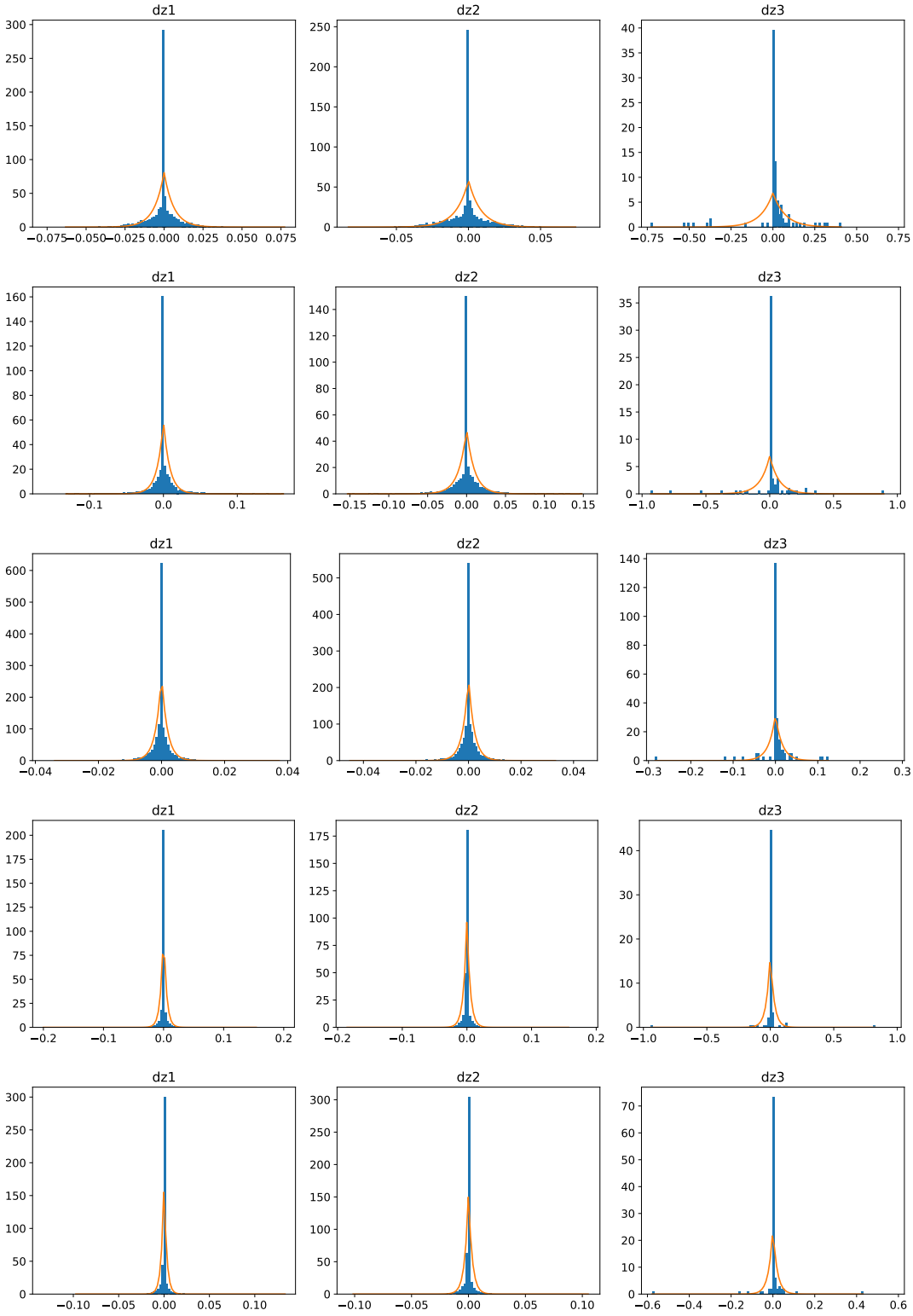

Figure C.2: Pre-activation gradient distributions at different snapshots during training of a fully-connected ReLU-activated neural network with two hidden layers (500 neurons each) on the MNIST image classification dataset.

C.3 QBP Error Statistics

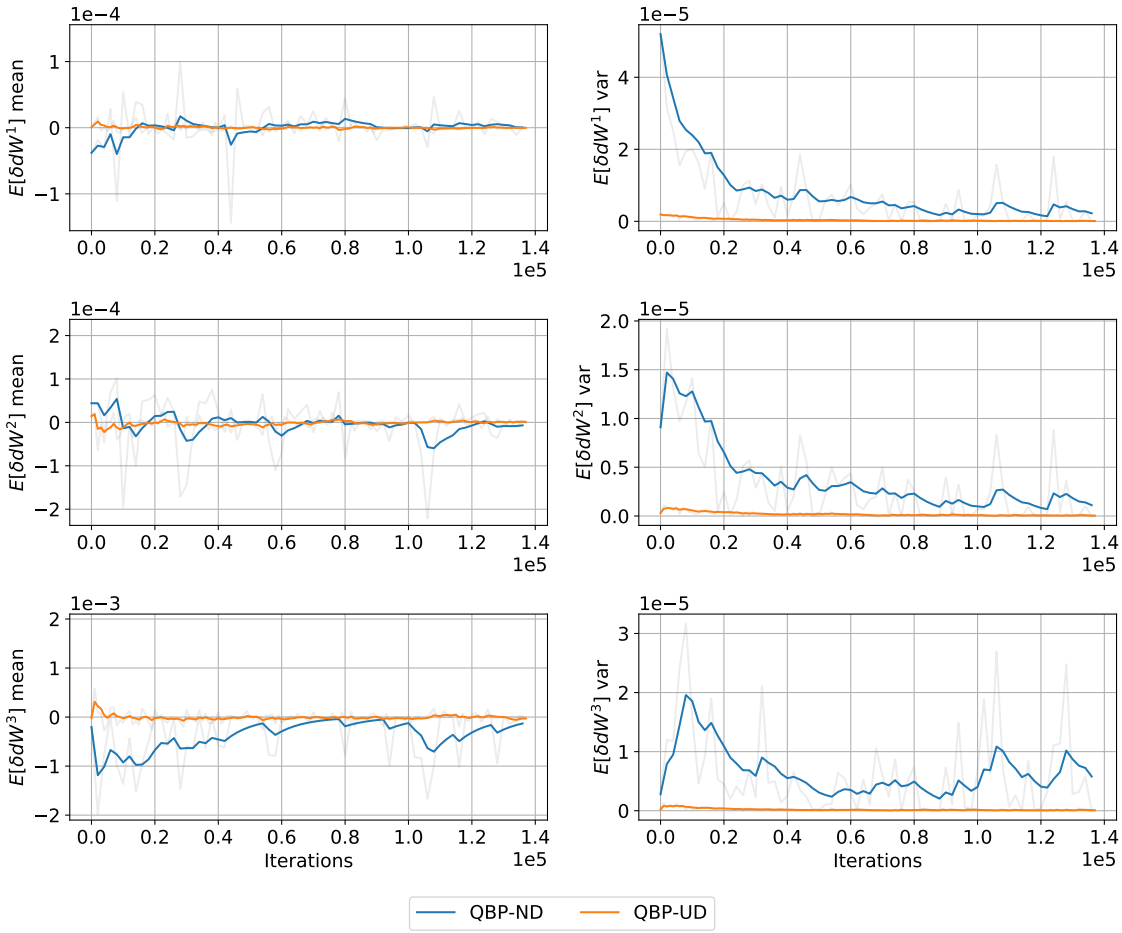

Figure C.3: Statistics for mean of weight gradient error when using QBP with constant step size $\Delta^l = 0.1$ and either no dither or uniform dither.

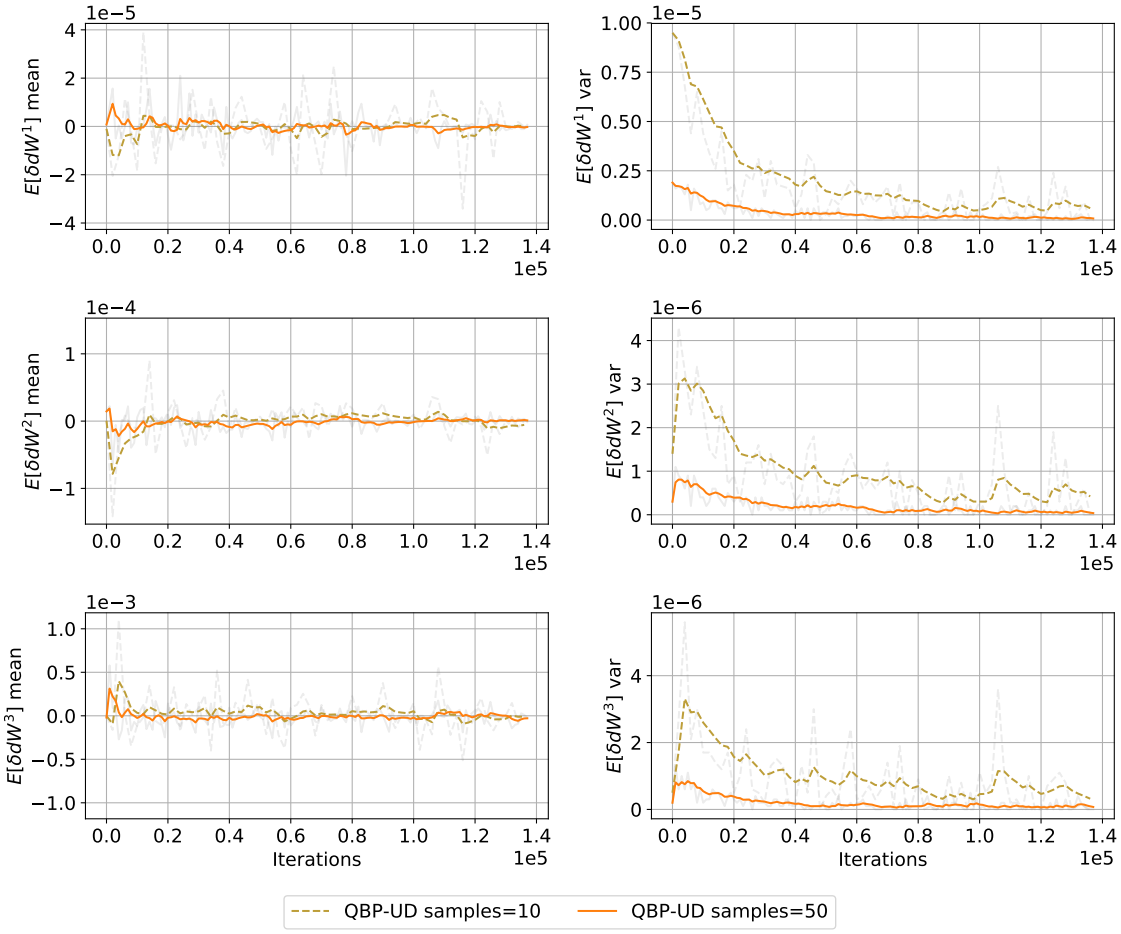

Figure C.4: Statistics for mean of weight gradient error when using QBP with constant step size $\Delta^l = 0.1$, uniform dither and either using 10 error samples or 50 error samples.