Improving the Accuracy of the Winograd Convolution for Deep Neural Networks

by

Barbara Barabasz

Dissertation

Submitted to the School of Computer Science and Statistics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
(Computer Science)

School of Computer Science and Statistics

TRINITY COLLEGE DUBLIN

March 2022

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work.

.....

Barbara Barabasz

Dated: March, 2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this Dissertation upon request.

.....

Barbara Barabasz

Dated: March, 2022

Abstract

This thesis is focused on Winograd's fast algorithms calculating discrete convolutions. More precisely, on the most optimal in terms of the number of operations subclass of them, that is Toom-Cook algorithms. These algorithms reduce the number of operations in a calculation process from $O(n^2)$ to $O(n)$. That means the larger input data is taken the bigger reduction in number of operations we have and in a consequence the greater speed up is obtained. However, the accuracy of computations declines as the input size increases. We undertake this dichotomy in our thesis. The problem is very essential because calculating discrete convolutions is a one of the main bottleneck in convolutional neural networks (CNNs) computations. The time required to train and use CNNs is challenging even for modern computers.

In our thesis we formulate algorithms to construct transformation matrices used in Toom-Cook convolutional algorithms. After analysing the mathematical background of these algorithms we prove that the error of Toom-Cook convolution computations grows exponentially with the input data size. This is caused by poor numerical properties of the transformation matrices. We also demonstrate that the error of the so called modified Toom-Cook algorithm is smaller than the error of the original one although it still grows exponentially. Then we identify the components of a numerical error of these convolution algorithms.

We propose some techniques that improve the accuracy of the Toom-Cook convolution computations and verify them empirically for the wide range of input data sizes. We find that we can tune not only input data size but also the number, degree and root points of polynomials used to construct transformation matrices (real valued Vandermonde matrices).

We investigate ways of choosing root points of polynomials which determine properties of the transformation matrices. We construct sets of optimal root points for various input tile sizes both in a single and a mixed precision.

We find that the Chebyshev points which work very good for bigger sizes real valued Vandermonde matrices do not work properly for smaller sizes used in DNNs. We propose and test the canonical summation order, based on the idea of Huffman coding, to further reduce the error in the dot product computations. All proposed methods result in about 50% reduction in floating point error of the final computations.

Finally, we come back to the general case of Winograd algorithms. We present the algorithm for the construction of Winograd transformation matrices. We present a method, using superlinear polynomials, which improves floating point accuracy. The presented version of the Winograd algorithm offers a larger space of trade-offs between computation complexity and accuracy using higher order polynomials. Thus, it allows us to find an attractive solutions that are not available when using Toom-Cook algorithms.

Acknowledgements

This thesis would not have been completed without the help and support of many people.

First and foremost I would like to thank my supervisor Prof. David Gregg for the opportunity and the intriguing topic.

Then I would like to thank Science Foundation Ireland which funded my study.

During the last three years I had an opportunity to cooperate with a few excellent researchers and I would like to thank specifically Prof. Kirk M. Soodhalter for his stimulating discussions and explanations. I am also very grateful to anonymous reviewers from ACM TOMS Journal for their deep and constructive comments, which were very helpful for my work.

During my study I had a great opportunity to work for Arm Research in Cambridge, UK. I would like to thank Matthew Mattina and Ganesh Dasika for giving me that chance. My special thanks are due to my manager at Arm, Andrew Mundy, who was always willing to listen to my ideas and did not try to persuade me to do things his way. I would like to thank him and others I have met there for creating an inspiring working environment.

I would also like to thank my lab mates: Kaveena, James, Andrew, Yuan and Asad and others for sharing the office for three years. They made my study an interesting and enjoyable experience.

Finally, I would like to thank my family and friends who supported and motivated me in this project. I am utterly grateful to my mom and dad, for motivating me.

My greatest gratitude is to my husband Grzesiek and son Kuba. Their unending love and belief in me kept me on track through last three years. Thanks for their countless talks about my work, their patience, understanding and all kinds of support I got from them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is focused on improving the numerical accuracy of fast convolution algorithms - Toom-Cook/Winograd algorithms - used generally for deep neural networks (DNNs) and especially for one of DNNs subclass, namely for convolutional neural networks (CNNs). The wide range of CNNs applications (for example image recognition (Antipov, Berrani, and Dugelay 2016), natural language processing (Wang and Gang 2018) and forecasting (Vijh et al. 2021)) require very fast - often real time computations and simultaneously with a high accuracy of numerical calculations. These needs and expectations convince to scientific research of fast and accurate algorithms for convolution computations. A firm conclusion that it is possible to achieve these goals is at the basis of my choice of the subject of the dissertation. However, the main goal of my thesis focused on increasing accuracy of convolution calculations without increasing the time of computations.

Winograd/Toom-Cook algorithms are used mostly in convolution computations according to the fact, that they have the least time complexity (number of multiplication operations) compared to other algorithms that can be used in these calculations. These algorithms focus attention (He et al. 2015; Taigman et al. 2014; Dean et al. 2012; Le et al. 2012; Liu et al. 2018) and are implemented in standard libraries of many giant companies, for example Facebook, Arm, Intel. The main idea of these algorithms is to transform input data and weights, perform convolution computations and to transform them back. The

matrices required for transformation are called Vandermonde matrices and are constructed from chosen parameters (points). After transformation convolution computations requires a less number of multiplication operations. The complexity decreases from $O(n^2)$ to $O(n)$. The additional transformation complexity amortizes over multiple use of the same sets of weights and input tiles. Winograd/Toom-Cook algorithms perform calculations of a convolution using a variable number of operations depending on the number of parameters (root points). Namely, the number of choosen root points determines the number of columns in transformation matrices and subsequently the number of multiplication operations.

Unfortunately, the less number of operations is needed the bigger numerical error of the result is created. The error grows exponentially with the linear reduction in number of operations. That is because of inadequate numerical properties of the Vandermonde matrices with elements from the field of real numbers. This error propagates further through all these computations, it is accumulated and thus the final result of computations is seriously affected by it. Hence the final result may be quite distant from the established or expected theoretical one and subsequently cause a low performance of the entire network. For example an accumulated numerical error results in wrong features extraction that finally gives incorrect image classification.

For the sake of completeness of the research, it is worth presenting the basic properties of the artificial neural networks. Artificial neural networks (ANNs) are biologically inspired mathematical models simulating the human brain. It is commonly accepted that the origins of ANNs date back to the 1940s as McCulloch and Pitts then proposed the first computational model of a biological neuron. ANNs possess the ability to learn from data, and the process of learning is known as the training process.

Deep neural networks (DNNs) create a subclass of the ANNs and consist of multiple hidden layers that extract input features.

The results of Toom-Cook/Winograd algorithms can be used in CNNs, a subclass of DNNs as was mentioned above. CNNs consist typically of mul-

tiple layers of various types - convolutional layers, pooling layers and fully-connected layers. Some of them are controlled by latent parameters, which in general are weights that are learnt during a training process. The convolutional layers perform convolutions to extract features from the inputs and produce the feature maps. A pooling layer reduces the dimension of a feature map while preserving the spatial locality of the features in the feature map. Fully-connected layers perform classification and regression. It is possible to train these layers to achieve a high accuracy (over 80%) for various tasks, especially in visual pattern recognition problems. However, even using modern computers, CNNs are still time and memory consuming. Up to 95% of convolutional neural network computational time is spent on computations of convolution (He and Sun 2015). For this reason speeding up convolution computations is very significant problem in CNNs. The computational time depends, among the others, on the number of executed operation. The fastest convolutional algorithms in terms of number of multiplications are the family of Toom-Cook/Winograd convolution algorithms. However, the flaw of these algorithms is poor accuracy of the computations. For this reason improving the numerical accuracy of Toom-Cook/Winograd algorithms is so important.

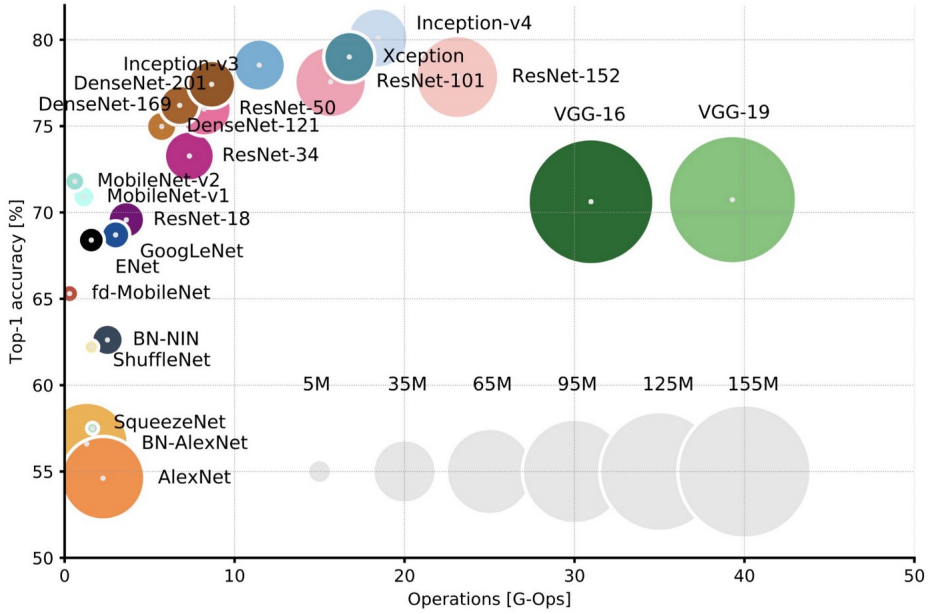


Figure 1-1: $Top-1$ accuracy versus number of operations and parameters for a chosen neural network (Canziani, Paszke, and Culurciello 2016)

In Figure 1-1 several convolutional networks are presented with the emphasis on their computational accuracy, number of parameters and executed operations. The current research efforts are focused on reducing the number of operations (x-axis) and on improving the accuracy (y-axis). That means to achieve for a network a position in the top left corner of the presented figure. In this thesis we present our theoretical and experimental results that allow to achieve these goals. The thesis aiming is at improving the numerical accuracy of fast convolution algorithms and placing networks in the top left corner of the chart Figure1-1.

1.1 General Research questions

This thesis answers the question, if it is possible to improve the accuracy of the Toom-Cook/Winograd convolution computations. More precisely, the main objective of the thesis is to detect the sources of floating point error, then to find and propose methods for reducing numerical error in fast convolution computations to keep the proper accuracy of the final result. We focus on the most optimal in terms of time complexity algorithms that is Toom-Cook/Winograd fast convolution algorithms.

To reach this aim, the following specific objectives and questions are posed.

- Analysing and developing the algorithms to construct transformation matrices that are involved in Toom-Cook/Winograd algorithms computations,
- Examining Toom-Cook/Winograd algorithms in terms of which parameters should be tuned to achieve speeding up and a proper accuracy of the algorithms,
- What factors have an impact for the accuracy of Toom-Cook/Winograd convolution computations?
- Are there any methods to decrease each component of the numerical error in Toom-Cook/Winograd convolution computations?

- Can we balance the trade-offs of Toom-Cook/Winograd algorithm time complexity and accuracy of convolution computations?

1.2 Contributions

This thesis is focused on improving the floating point accuracy of the Toom-Cook/Winograd fast convolution algorithms and speeding them up. Accordingly, our main contributions to the numerical and algorithmic fields and DNNs as well, are the following:

- We provide the exact algorithm with pseudocode and mathematical formula for constructing elements of transformation matrices used in Toom-Cook/ Winograd algorithm in dependency on chosen parameters (root points) (Section 3.1, Section 3.1.1, Algorithm 5, Section 3.2, Algorithm 6).

- We analyse the worst-case floating point normwise and componentwise error bounds in the Toom-Cook algorithm and show that the error grows exponentially as an error from transformations with Vandermonde matrices dominates overall accuracy(Section 4.4, Theorem 5, Theorem 6).

- We present a formal analysis of the normwise and componentwise floating point error bound for the 'modified' Toom-Cook algorithm, and prove that its error is lower then in the original algorithm, but still exponential (Section 4.8, Theorem 9, Section 4.9).

- We demonstrate that the order of evaluation of floating point expressions in the linear transformation impacts accuracy. We propose a canonical Huffman tree evaluation order that reduces the average error at no additional cost in computations (Section 5.2.1). The main idea of it is to choose elements of the similar order of magnitude for every step, to do the calculations.

- We experimentally evaluate strategies for selecting algorithm coefficients for typical Deep Neural Network convolutions. We identify relationships

between parameters (points) used to construct Vandermonde matrices which improve the accuracy of convolution computations (Section 5.2.2).

- We investigate mixed-precision algorithms that compute the transformations in a double precision. These methods reduce the error typically by around one third in our experiments (Section 5.5).

- We provide the exact algorithm with pseudocode and formula for constructing the elements of transformation matrices for the Winograd algorithm with higher degree polynomials in dependency on chosen parameters (points) (Section 6.1.1, Algorithm 7, Algorithm 8).

- We present experimental results for Winograd convolution algorithm and show that using a higher-order polynomials can reduce the floating point error (Section 6.3).

- We point out the trade-off between computational complexity and floating point accuracy for Toom-Cook and Winograd algorithms (Section 6.2).

- We experimentally identify cases where using the Winograd algorithm with a higher-order polynomials can improve the accuracy of image recognition and/or reduce the complexity of computations when using a lower precisions (Section 6.3.2).

- We share an implementation of Toom-Cook and Winograd transformation matrices construction and convolution algorithms on GitHub .

Results presented in this thesis are based on the following publications:

- "Error Analysis and Improving the Accuracy of Winograd Convolution for Deep Neural Networks", Barbara Barabasz, Andrew Anderson, Kirk M. Soodhalter and David Gregg, ACM Transactions on Mathematical Software (TOMS), 2020, vol. 46, pp 1 - 33 (Barabasz et al. 2020)

- "Winograd Convolution for DNNs: Beyond Linear Polynomials", Barbara Barabasz and David Gregg", AI*IA 2019 - Advances in Artificial Intelli-

gence - XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19-22, 2019, Proceedings, Lecture Notes in Computer Science, 2019 vol. 11946, pp307–320", (Barabasz and Gregg 2019)

- "Hardware and software performance in deep learning" in "Many-Core Computing: Hardware and Software", Andrew Anderson, James Garland, Yuan Wen, Barbara Barabasz, Kaveena Persand, Aravind Vasudevan and David Gregg, 2019 pp. 141-161 (Anderson et al. 2019)

1.3 Thesis structure

After the introduction given in Chapter 1, in Chapter 2 we present the state-of-the-art and also identify current unresolved issues and problems.

At the beginning of the Chapter 2 we present a brief history of Convolutional Neural Networks. Next, we describe several state-of-the-art net architectures. Then, we list CNNs layers and describe functions and operations they perform. We introduce the IEEE 754 floating point number system – a method of coding real values in modern computing on variable number of bits and define the floating point error of a representation and operations. Finally, we discuss direct and fast convolution algorithms in the context of their accuracy and time complexity.

In Chapter 3 we present a theoretical background and detailed description of the Winograd/Toom-Cook convolution algorithm and its modified version. We provide the algorithms to construct transformation matrices for any chosen parameters. In particular Algorithm 5, Algorithm 6 are novelties in the field.

In Chapter 4 we focus on theoretical properties of the Winograd/Toom-Cook algorithm and its modified version. At the beginning we provide the necessary mathematical background and references to literature. Then we present the numerical error boundaries for one- and two-dimensional Winograd/Toom-Cook convolution computations and identify the error components.

In Chapter 5 we propose several practical methods to improve the accuracy of the Winograd/ Toom-Cook computations. We present the methodology and the results of tests executed on random data. At the end we discuss our results and compare them with the state-of-the-art.

In Chapter 6 we focus on Winograd convolution algorithms with super linear polynomials. We formulate an algorithm to construct transformation matrices. We also discuss its time complexity and compare it to the optimal Toom-Cook/Winograd convolution algorithm. We present the accuracy test results for various versions of the Toom-Cook and Winograd algorithms for random data. We also provide an example results of the whole network accuracy for a subset of ImageNet dataset with data stored in lower precisions.

Finally, in Chapter 7, we summarise and discuss all the results presented in the thesis and suggest some possible future research directions.

Chapter 2

Background and Literature Review

In this chapter we give a brief description of several background topics and introduce basic notions, definitions and notations that are used throughout this thesis. In what follows we give in Section 2.1 a brief introduction to convolutional neural networks (CNNs) describing layers that CNNs consist of with special emphasis to convolutional layer. We fix basic mathematical notions, definitions and notations in Section 2.2. Notational conventions connected with Winograd algorithm are in Section 2.2.2. Next, in Section 2.3 we present concepts of a floating point number system and its impact on numerical errors of number representations and computations. Later, in Section 2.4 we provide definitions of convolution and a direct/naive convolution algorithm. Section 2.5 starts with the discussion of the idea of well known fast convolution algorithms, their time complexity and computational accuracy. Finally, we give a description of Toom- Cook and Winograd family of algorithms and highlight the current unresolved issues and problems in the literature of the subject.

2.1 Convolutional Neural Networks

Artificial Neural Networks (ANNs) are inspired by the biological neural networks. They have the ability to learn by processing data. This learning process, called also a training process, is such an adaptation of the network parameters which leads to the better performance of the tasks. More precisely, during a

training process after weight initialization, an output is calculated and then an error is fixed. Then, weights are updated accordingly to the values of errors, to obtain a more accurate performance of the task.

ANNs consist of artificial neurons and connections between them. The neurons take an input value and process it using internal parameters to produce an output. Typically they are organized in layers that perform different kinds of transformations (see Figure 2-1). The output of a one layer becomes the input to the next one. The idea of applying mathematics and logic to create a computational model of a neuron or, more generally, a model of a neural network was introduced by McCulloch & Pitt in 1943 (Mcculloch and Pitts 1943).

The origins of CNNs are usually set in the 1990s and related to the paper "Gradient-Based Learning Applied to Document Recognition" by Y. LeCun et al. The authors demonstrated that a CNN model can be successfully applied for handwritten characters recognition (LeCun et al. 1989). However, there were some earlier works on convolutional neural networks. For the sake of completeness we mention a multilayer neural network - Neocognitron proposed in 1979 by K.Fukushima to recognise Japanese handwritten characters and to pattern recognition (Fukushima 1980). Their work was inspired by Hubel and Wiesel hierarchical model of the visual nervous system (Hubel and Wiesel 1959). In the 1980's a group of scientists from Carnegie Mellon University - Waibel, Lang, Hinton and others - proposed time-delay neural networks and applied them successfully to speech recognition (Waibel et al. 1989; Lang 1988; Lang, Waibel, and Hinton 1990).

Deep Neural Networks (DNNs) use many layers to extract multiple levels of features. They consist of input and output layers as well as the subsequence of multiple hidden layers between them (see Figure 2-1) — their number is called the network depth. Each layer performs a linear or a non-linear mathematical operation on an input and it is responsible for extracting a different features.

Convolutional Neural Networks (CNNs) create a class of deep neural networks. In these networks, hidden layers perform a linear operation defined as

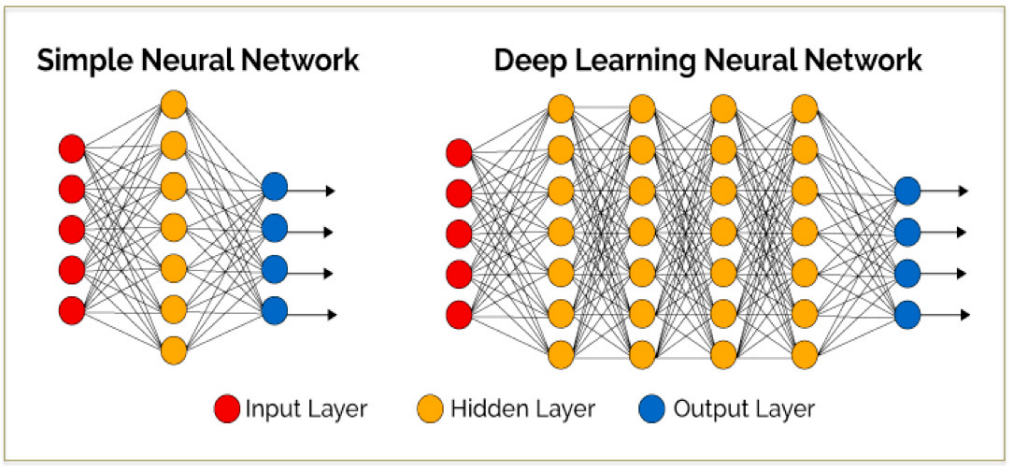


Figure 2-1: Artificial Neural Network and Deep Neural Network (figure taken from blog (McGinty 2018))

convolution between the input and the kernel. [1]. For every layer the same set of weights (kernel) is applied several times to the input regions. The layer parameters are designed to extract particular features of the input data. They are used mostly in computer vision, image recognition and object detection where the input is represented as an array of pixels.

Convolutional Neural Networks in the form as we know and use them now, were presented by Yann LeCun in 1989 (Backpropagation Applied to Handwritten Zip Code Recognition (LeCun et al. 1989)). He used a backpropagation algorithm to learn convolution weights from images of hand-written numbers. However, the widespread adoption of this approach started in 2012 with AlexNet proposed by Krizhevsky (Krizhevsky, Sutskever, and Hinton 2012).

2.1.1 ImageNet and ILSVRC

ImageNet is a set of images tailored for a computer vision research. The work on this dataset project started in 2006 and was led by Fei-Fei Li (Deng et al. 2009). Their goal was to improve the training data for image classification and object detection. Right now, it is the most popular benchmark for validating DNN accuracy. Other common used banchmark datasets for an image recogni-

[1] Formally the actual mathematical operation is called a correlation but in machine learning field called a convolution, by a convience

tion are for example MNIST (LeCun, Cortes, and C.J.Burges 2010) and CIFAR10 (Krizhevsky 2009). The generally accepted way of network accuracy measurement is $top-1$ and $top-5$ scores. The first one is the ratio of the number of images recognized correctly by network and the number of all tested images. The second one is the ratio of the number of images for which the correct label is in five labels highest scores by the network and the number of all tested images.

The ImageNet Large Scale Visual Recognition Challenge - ILSVRC is an annual competition that focuses around developing image recognition techniques and algorithms. The goal is to achieve the highest accuracy that is the highest percentage of correctly classified images. The participants evaluate their algorithms/networks on a given dataset (subset of ImageNet) with 1000 image categories (labels). In 2012 AlexNet proposed by Krizhevsky (Krizhevsky, Sutskever, and Hinton 2012) won this competition, significantly improving the previous year result ($top-5$ score equal to 84.7% comparing to around 75% in 2011). It started a rapid development in the field of convolutional networks and winning models began to establish the state-of-the-art in the machine learning field.

2.1.2 Convolutional layer

Convolutional networks consist of a number of hidden layers (convolutional, activation and pooling layers), which perform various mathematical operations. They are applied to local regions of an input (see Figure 2-2). The last, fully-connected layer combines all of the outputs from the previous layer. Although CNNs are applied to various problems in this thesis, for simplicity we use the notations characteristic for an image classification.

Convolution computations (described in details in section 2.4) performed in convolutional layers are the most time consuming parts of the convolutional networks. Up to 95% of the whole time of an operating time of convolutional networks is spent on convolution computations (He and Sun 2015). Every convolutional layer performs a linear operation with its own set of parameters (set

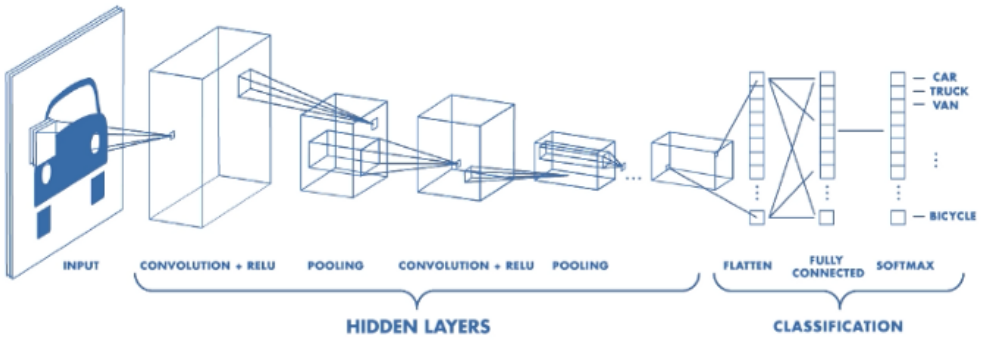


Figure 2-2: Convolutional Neural Network (source: uk.mathworks.com)

of weights called the kernel) over local regions of the input. The kernel with defined weights is moved over the input, applied to a few pixels at a time and a convolution of kernel and input data is computed. So, it means that the same kernel is used multiple times to compute the whole layer output. Also the same input data could be convolved with various kernels.

To avoid the reduction of the input size, the border of the zero values is added to the input. This operation is called padding. A stride determines how far a kernel moves over the input. If the stride is smaller than the kernel size we have so called overlaping. That means that the same values of the input are used more than once for different local convolution computations – Figure 2-3.

The input to a convolutional layer is a three dimensional matrix ($input\_height \times input\_width \times input\_channels$). A set of kernels consists of weights and biases. Those parameters are incrementally modified when performing a training. Weights are organized in a four dimensional matrix ($kernel\_height \times kernel\_width \times input\_channels \times output\_channels$). The most common sizes of the kernel in widely used CNNs are small, like 3×3 or 5×5.

The corresponding results of convolution computations for every input channel are summed up and a bias is added to produce a matrix referred to as a feature map. When we have multiple output channels we use the same input several times with different set of weights for every single output channel. The output of the convolutional layer is then a stack of such feature maps.

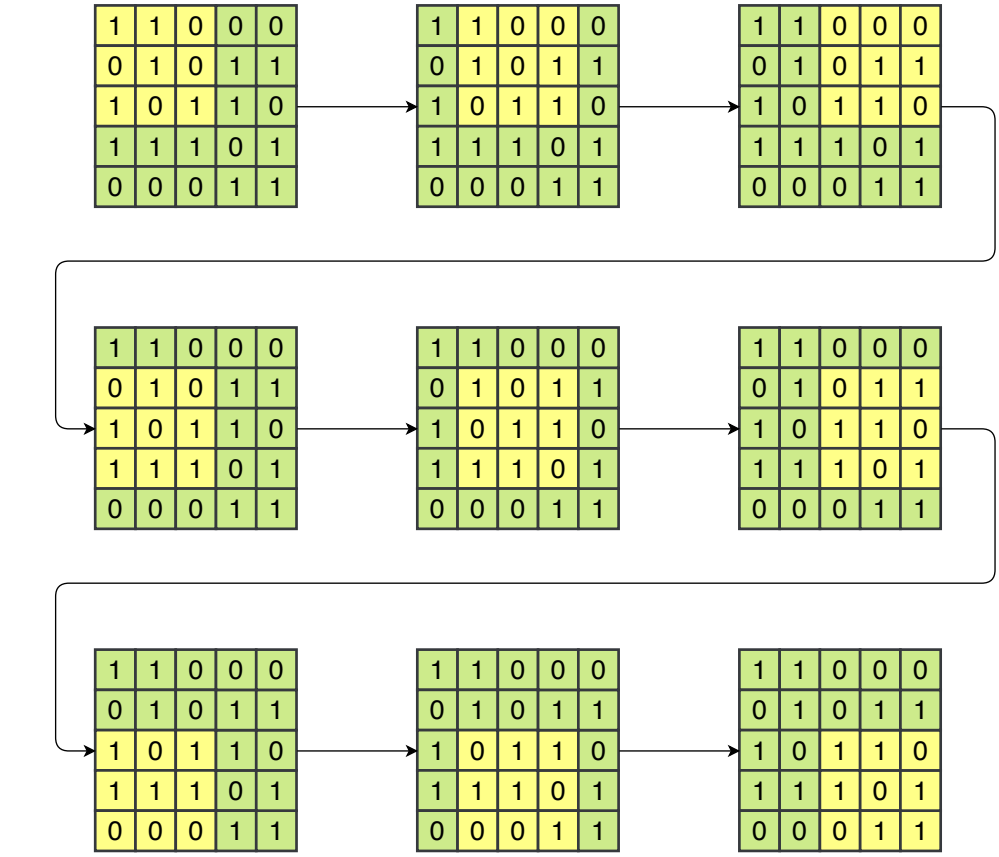


Figure 2-3: Overlapping during convolution.

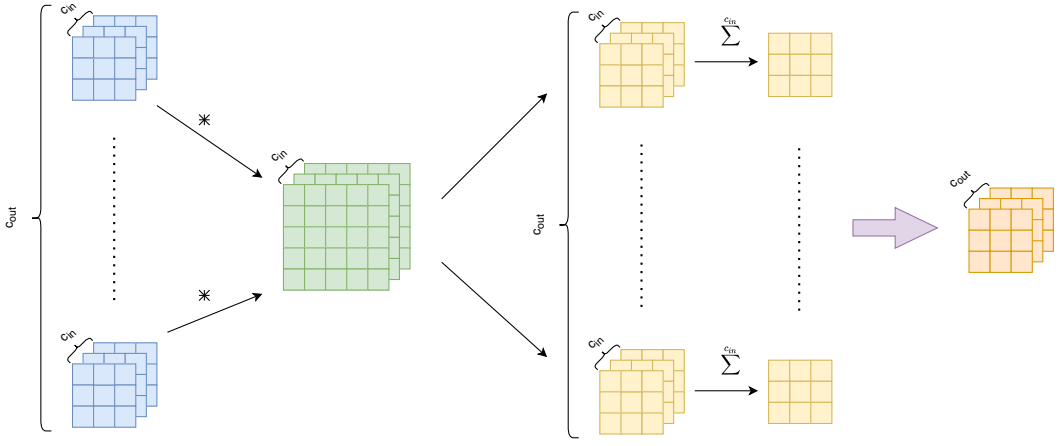


Figure 2-4: Convolution layer

2.2 Mathematical preliminaries and notations

In this section, we briefly introduce some mathematical preliminaries intensively used across this thesis. Definitions and notations presented here follow

Higham (Higham 2002), Datta (Datta 2010), Golub (Golub and Loan 2013) and Wilkinson (Wilkinson 1994) notation. Deeper mathematical explanations could be found in (Rudin 1986). Introduced definition and theorems are used mostly in Chapter 4 in theoretical error estimations. Some of them are also briefly recalled in the later parts of this thesis.

2.2.1 Notation

- $a, b, x,\ R$ - lower and capital letters denote scalars, constants, variables, and coefficients.

- $\mathbf{w} = [w_1, \ldots, w_n]$, $w(a) = \sum_i w_i a^{n-i}$ - vectors are denoted by bold lower letters, while corresponding polynomials by lower letter with an argument. For the clarity - polynomials in Chinese Remainder Theorem are denoted by capital letters $M(a)$ or $M_i(a)$.

- $|\mathbf{w}| = [|w_1|, \ldots, |w_n|]$ is a vector whose coefficients are equal to the absolute values of the vector $\mathbf{w}$ coefficients.

- $\mathbf{W} = [w_{ij}]$, $\mathbf{W}(a, b) = \sum_i \sum_j w_{ij} a^{i-1} b^{j-1}$ bold capital letters stand for matrices, while corresponding polynomials by bold capital letters with arguments.

$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{bmatrix}$$

- $\mathbf{W}_{i:}$ and $\mathbf{W}_{:j}$ denote the ith row and jth column of the matrix $\mathbf{W}$ respectively.

- $|\mathbf{W}|$ is a matrix which entries are equal to the absolute values of the matrix $\mathbf{W}$ elements .

$$|\mathbf{W}| = \begin{bmatrix} |w_{11}| & \cdots & |w_{1n}| \\ \vdots & \ddots & \vdots \\ |w_{m1}| & \cdots & |w_{mn}| \end{bmatrix}$$

- The special kind of matrices involved in Winograd algorithm called Vandermonde matrices will be denoted by $\mathbf{V}$ unless there will be a need to differentiate between them. These matrices has a special form of entries $v_{ij} = a_i^{j-1}$ for $i = 1, \ldots, m$ and for $j = 1, \ldots, n$.

$$\mathbf{V} = \begin{bmatrix} 1 & a_1 & \cdots & a_1^{n-1} \\ 1 & \vdots & \ddots & \vdots \\ 1 & a_m & \cdots & a_m^{n-1} \end{bmatrix}$$

- Field of real numbers is denoted by $\mathbb{R}$, while $\mathbb{F}$ stands for a field in the general case.

- Ring of polynomials of a one variable over a field $\mathbb{F}$ is denoted by $\mathbb{F}[a]$.

- F denotes a set of floating point numbers. This set consists of real values that can be represented exactly in a finite floating point representation, $F \subset \mathbb{R}$.

- A function $fl : \mathbb{R} \to F$ returns a floating point representation of a real value.

- The floating point representation error is bounded by ε - called a machine epsilon

- Computational result that includes floating point error is denoted with hat $\hat{\mathrm{s}}$, while exact (theoretical) solution is denoted s.

- $f(a)$ - functions are denoted by lower letters with argument.

- Monic polynomial means any polynomial with the leading coefficient equal to 1.

- $\mathbf{w}(a) \mod \mathbf{x}(a)$ is equal to a polynomial which is the remainder of the polynomials division $\mathbf{w}(a)/\mathbf{x}(a)$. In algorithms we also use the notation $R_{\mathbf{x}(a)}[\mathbf{w}(a)]$.

- $GCD(\mathbf{w}(a), \mathbf{x}(a))$ stands for the greatest common divisor of the polynomials $\mathbf{w}(a)$ and $\mathbf{x}(a)$.

- To estimate and compare a time complexity of algorithms in a consistent way we use $O(\cdot)$ notation (Knuth 1998). It describes the asymptotical behaviour of a function. The equality $f(n) = O(g(n))$ holds if and only if there exist a positive integers r, n_0 such that $f(n) \leq rg(n)$ for all positive integers $n \geq n_0$.

- α, β, γ - greek letters that show how the exact number of particular operations change with the size of data - n. For example $\alpha^{(n)} = n + 1$

- P_i stands for a set containing exactly i values.

- $\mathrm{vec}(\mathbf{W})$ function that constructs a vector from a matrix $\mathbf{W}$ by stacking all its columns into a one column.

- $\mathrm{Diag}(\mathbf{w})$ function that constructs a square diagonal matrix $\mathbf{W}$ from a vector $\mathbf{w}$ by putting its elements on the main diagonal.

- $\mathbf{W}^T$ stands for a transposition of a matrix $\mathbf{W}$.

- $\mathbf{W}^{-1}$ stands for an inverse of a matrix $\mathbf{W}$.

2.2.2 Toom-Cook/Winograd algorithm notation

- k,$k \times k$, kernel size - dimension of the space which kernel vector/matrix belongs to.

- m,$m \times m$, output size - dimension of the space which convolution algorithm resultant vector/matrix belongs to.

- n,$n \times n$, input size - dimension of the space which input data vector/matrix belongs to.

- $\mathbf{A}, \mathbf{B}, \mathbf{G}$ - matrices used to perform Toom-Cook/Winograd algorithm.

- $\mathbf{G}^{\mathbf{modify}(n)}, \mathbf{A}^{\mathbf{modify}(n)}, \mathbf{B}^{\mathbf{modify}(n)}$ - matrices used for a modified version of Toom-Cook/Winograd algorithm with input data of the size $n/n \times n$.

- $\mathbf{A}^{(n)}, \mathbf{G}^{(n)}, \mathbf{B}^{(n)}$ - matrices used for Toom-Cook/Winograd algorithm with input data of the size $n/n \times n$.

- $\mathbf{A}^{\mathbf{T-C}}, \mathbf{B}^{\mathbf{T-C}}, \mathbf{G}^{\mathbf{T-C}}$ - matrices used for the most optimal (Toom-Cook) version of the Winograd algorithm.

- $\mathbf{A}^{\mathbf{W}}, \mathbf{B}^{\mathbf{W}}, \mathbf{G}^{\mathbf{W}}$ - matrices used for any (except the most optimal one) version of the Winograd algorithm.

In Chapter 6 we consider only modified versions of Toom-Cook and Winograd algorithms and use the notation introduced by Winograd (Winograd 1980b) and followed by Lavin (Lavin and Gray 2016)

- $F(m, k)$ - set of Winograd algorithms with a kernel of the size k and the output size equal to m.

- $F_{T-C}(m, k)$ - the most optimal version of the Winograd algorithm (Toom-Cook) with the kernel of the size k and the size of the output m.

- $F_W(m, k)$ - set of all Winograd algorithms (with the exception of the most optimal one) with the kernel of the size k and the size of the output m.

2.2.3 Basic definitions

Definition 1. *The Hadamard product of vectors (element-wise multiplication) is denoted by* $\odot$. *For any vectors* $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ *it is defined as* $\mathbf{a} \odot \mathbf{b} = \mathbf{c} \in \mathbb{R}^n$

$$\textit{where} \quad \mathbf{c} = [a_1 b_1, \ldots, a_n b_n]$$

For any matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times m}$ *it is defined as* $\mathbf{A} \odot \mathbf{B} = \mathbf{C} \in \mathbb{R}^{n \times m}$

$$\textit{where} \quad c_{ij} = a_{ij} b_{ij} \quad \forall i = 1, \ldots, n \quad \forall j = 1, \ldots, m.$$

Definition 2. *The Kronecker product of matrices is denoted by* $\otimes$.
For any matrices $\mathbf{A} \in \mathbb{R}^{n\times m}$ *and* $\mathbf{B} \in \mathbb{R}^{p\times q}$

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{C} \in \mathbb{R}^{np\times mq}$$

$$\textit{where} \quad \mathbf{C} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n1}\mathbf{B} & \dots & a_{nm}\mathbf{B} \end{bmatrix}$$

Definition 3. *Let* $\mathbf{C}, \mathbf{E} \in \mathbb{R}^{n\times m}$ *be block matrices where each block is of the size* $s_i \times s_j$
The Khatri-Rao product is the blockwise Kronecker product defined by $\mathbf{C} \otimes_{KR} \mathbf{E} = \left[(\mathbf{C_{ij}} \otimes \mathbf{E_{ij}})_{ij}\right]$ *for* $i = 1, \dots, n/s_i$ *and* $j = 1, \dots, m/s_j$. *That is*

$$\begin{bmatrix} \mathbf{C_{11}} \otimes \mathbf{E_{11}} & \dots & \mathbf{C_{1m/s_j}} \otimes \mathbf{E_{1m/s_j}} \\ \vdots & \ddots & \vdots \\ \mathbf{C_{n/s_i1}} \otimes \mathbf{E_{n/s_i1}} & \dots & \mathbf{C_{n/s_im/s_j}} \otimes \mathbf{E_{n/s_im/s_j}} \end{bmatrix}$$

Definition 4. *Operation* $*$ *stands for a convolution.*
For two continuous functions f *and* g *convolution over a finite range* $(0, t)$ *is given by the formula*

$$(f * g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau$$

Discrete convolution is defined over $\mathbb{Z}$ *- set of all integers, for* f *and* g *discrete functions.*

$$(f * g)(n) = \sum_i f(i)g(n-i)$$

Definition 5. *Jacobian of a differentiable function* $f : \mathbb{R}^n \to \mathbb{R}^m$ *is a matrix* $\mathbf{J}$ *which elements are equal to the first order partial derivatives.*

$$\mathbf{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial fm}{\partial x_n} \end{bmatrix}$$

2.2.4 Norms

In general, a norm can be defined in any vector space. However, in this thesis we make use of finite vector spaces $\mathbf{R}^n$ and $\mathbf{R}^{n\times m}$. Thus we focus on vector and matrix norms and for simplicity we limit definitions to finite vector spaces.

Definition 6. *Vector norm*

A vector norm is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ *that transforms coefficients of a vector* $\mathbf{w} = [w_1, \ldots, w_n]$ *into a scalar (real number). This function has to satisfy the following three conditions for any vectors* $\mathbf{w} = [w_1, \ldots, w_n]$ *and* $\mathbf{v} = [v_1, \ldots, v_n] \in \mathbb{R}^n$

- $\|\mathbf{w}\| \geq 0$ *and* $\|\mathbf{w}\| = 0$ *if and only if* $w_i = 0 \quad \forall i = 1, \ldots, n.$
- $\|k\mathbf{w}\| = |k| \, \|\mathbf{w}\| \; \forall k \in \mathbb{R}$
- $\|\mathbf{w} + \mathbf{v}\| \leq \|\mathbf{w}\| + \|\mathbf{v}\|$ - *(the triangle inequality).*

In this thesis we use the following two vector norms:

$$\| \mathbf{w}\|_1 = \sum_{i=1}^{n} |w_i| \tag{2.1}$$

$$\|\mathbf{w}\|_2 = \sqrt{\sum_{i=1}^{n} |w_i|^2} \tag{2.2}$$

Definition 7. *Matrix norm*

The matrix norm is a function $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ *that transforms the elements of a matrix* $\mathbf{R} \in \mathbb{R}^{m\times n}$ *into a scalar (real number). This function has to satisfy the following conditions for any matrices* $\mathbf{W}, \mathbf{V} \in \mathbb{R}^{m\times n}$

- $\|\mathbf{W}\| \geq 0$ *and* $\|\mathbf{W}\| = 0$ *if and only if* $w_{ij} = 0 \quad \forall i = 1, ..., m \; , \; j = 1, ..., n$
- $\|k\mathbf{W}\| \leq |k| \, \|\mathbf{W}\| \; \forall k \in \mathbb{R}$
- $\|\mathbf{W} + \mathbf{V}\| \leq \|\mathbf{W}\| + \|\mathbf{V}\|$ *(the triangle inequality)*

There are some specific matrix norms corresponding to vector norms. Given a vector norm on $\mathbb{R}^n$, the corresponding subordinate (induced) matrix norm is defined below.

Definition 8. *Let $\|\mathbf{w}\|$ be a vector norm on $\mathbb{R}^n$. Induced matrix norm $\|\mathbf{W}\|$, where $\mathbf{W} \in \mathbb{R}^{m\times n}$ is defined by the formula*

$$\|\mathbf{W}\| = \max_{\mathbf{x}\neq 0} \frac{\|\mathbf{W}\mathbf{x}\|}{\|\mathbf{x}\|}$$

From the definition, it follows that all induced norms fulfill the following inequality

$$\|\mathbf{U}\mathbf{V}\| \leq \|\mathbf{U}\| \, \|\mathbf{V}\| \tag{2.3}$$

In this thesis we use the following matrix norms

$$\| \mathbf{W}\|_1 = \max_j \sum_i |w_{ij}| \quad \text{induced by} \quad \|\mathbf{x}\|_1 \tag{2.4}$$

$$\|\mathbf{W}\|_2 = \max_{\mathbf{x}} \frac{\|\mathbf{W}\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \quad \text{induced by} \quad \|\mathbf{x}\|_2 \tag{2.5}$$

$$\|\mathbf{W}\|_F = \sqrt{\sum_i \sum_j |w_{ij}|^2} \quad \text{- Frobenius norm.} \tag{2.6}$$

The Frobenius norm is often used in a numerical analysis instead of $\|\cdot\|_2$, because it is easy to compute and $\|\mathbf{A}\|_F = \| \, |\mathbf{A}| \, \|_F$, where $|\mathbf{A}|$ is a matrix with entries equal to the absolute values of the entries of $\mathbf{A}$ (Wilkinson 1994).

Definition 9. *Any two norms $\|\cdot\|$ and $\|\cdot\|'$ in a vector space are equivalent if and only if there exist real numbers r_1, r_2 such that for any vector $\mathbf{w}$, the following inequalities hold:*

$$r_1 \, \|\mathbf{w}\|' \leq \|\mathbf{w}\| \leq r_2 \, \|\mathbf{w}\|'$$

The definition for matrix norms equivalence is formulated analougously.

Definition 10. *Any two norms $\|\cdot\|$ and $\|\cdot\|'$ in a matrix space are equivalent if and only if there exist positive real numbers r_1, r_2 such that for any matrix $\mathbf{W}$,*

the following inequality holds:

$$r_1 \, \|\mathbf{W}\|' \leq \|\mathbf{W}\| \leq r_2 \, \|\mathbf{W}\|'$$

Definition 11. *Condition number of a function* $f(a) : \mathbb{R}^n \to \mathbb{R}^m$ *is defined as*

$$\kappa(x) = \limsup_{\epsilon \to 0} \{ \frac{\|f(y) - f(x)\|}{\|f(x)\|} / \frac{\|x - y\|}{\|x\|} \} \tag{2.7}$$

where $\|x - y\| \leq \epsilon$

If a function f *is differentiable then*

$$\kappa(x) = \frac{\|J_f(x)\|\|x\|}{\|f(x)\|} \tag{2.8}$$

2.2.5 Buniakowski-Schwarz inequality

In general, an inner product can be defined in any vector space. However, similarly as in the norm case, we limit our definitions to vector or matrix spaces $\mathbb{R}^n$ or $\mathbb{R}^{m \times n}$ defined over the field of real numbers $\mathbb{R}$.

Definition 12. *Inner product in* $\mathbb{R}^n$

An inner product in a vector space $\mathbb{R}^n$ *is a function*

$$\langle \ , \ \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

such that the following rules hold for any vectors $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^n$

- $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$.
- $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{u}, \mathbf{w} \rangle + \langle \mathbf{v}, \mathbf{w} \rangle$.
- $\langle k \cdot \mathbf{u}, \mathbf{v} \rangle = k \langle \mathbf{u}, \mathbf{v} \rangle$ *for any* $k \in \mathbb{R}$.
- $\langle \mathbf{u}, \mathbf{u} \rangle \ \geq 0$ *for any*
- $\langle \mathbf{u}, \mathbf{u} \rangle = 0$ *if and only if* $\mathbf{u} = \Theta$.

Definition 13. *Inner product in* $\mathbb{R}^{m \times n}$

An inner product in a vector space $\mathbb{R}^{m \times n}$ *(matrix space) is a function*

$$\langle \ , \ \rangle : \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} \to \mathbb{R}$$

such that the following conditions hold for any matrices $\mathbf{U}, \mathbf{V}, \mathbf{W} \in \mathbb{R}^{m \times n}$

- $\langle \mathbf{U}, \mathbf{V} \rangle = \angle \mathbf{V}, \mathbf{U} \angle$.
- $\langle \mathbf{U} + \mathbf{V}, \mathbf{W} \rangle = \langle \mathbf{U}, \mathbf{W} \rangle + \langle \mathbf{V}, \mathbf{W} \rangle$.
- $\langle k \cdot \mathbf{U}, \mathbf{V} \rangle = k \langle \mathbf{U}, \mathbf{V} \rangle$ *for any* $k \in \mathbb{R}$.
- $\langle \mathbf{U}, \mathbf{U} \rangle \ \geq 0$
- $\angle \mathbf{U}, \mathbf{U} \rangle = 0$ *if and only if* $\mathbf{U} = \mathbf{\Theta}$.

A vector space equipped with an inner product is referred to as an inner vector space. The defined above inner vector spaces are called Euclidean spaces.

Notice that just from the definition of an inner product it follows that

$$\langle \mathbf{w}, \mathbf{u} + \mathbf{v} \rangle = \langle \mathbf{w}, \mathbf{u} \rangle + \langle \mathbf{w}, \mathbf{v} \rangle$$

and

$$\langle \mathbf{u}, k \cdot \mathbf{v} \rangle = k \langle \mathbf{u}, \mathbf{v} \rangle$$

for any $k \in \mathbb{R}$ and $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^n$. Exactly the same equalities are true for matrices.

In Chapter 4 we use intensively Buniakowski-Schwarz inequality (see Introduction to functional analysis). This inequality is also known as Cauchy-Schwarz inequality, Schwarz inequality or Cauchy-Buniakowski-Schwarz inequality.

Theorem 1. *Buniakowski-Schwarz inequality*
Let $\mathbf{u}$ *and* $\mathbf{v}$ *be arbitrary vectors in an inner product space over the field of real numbers* $\mathbb{R}$. *Then*

$$|\langle \mathbf{u}, \mathbf{v} \rangle| \leq \|\mathbf{u}\| \, \|\mathbf{v}\| \tag{2.9}$$

where $\|\mathbf{u}\| = \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle}$ *and* $\|\mathbf{v}\| = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$

Our analysis in Chapter 4 is carried out in a real vector space $\mathbb{R}^n$ where the inner product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, $\mathbf{u} = [u_1, \ldots, u_n], \mathbf{v} = [v_1, \ldots, v_n]$ is defined as a dot product of $\mathbf{u}$ and $\mathbf{v}$.

Definition 14. *A dot product of two vectors* $\mathbf{u}$ *and* $\mathbf{v} \in \mathbb{R}^n$ *is defined as*

$$\mathbf{uv} = \mathbf{u}^T \mathbf{v} = \sum_{i=1}^{n} u_i \, v_i \tag{2.10}$$

2.3 Floating Point number system

In this section we describe the floating point IEEE 754 standard for various number of bits dedicated for every value. We also introduce notation and definitions for an estimation the floating point error that comes from a finite bits representation and arithmetic operations.

In this thesis we focus on floating point number system as FPU (Floating Point Unit) is widely used in modern devices for example Arm, Intel, (Reuther et al. 2020). There are also other number systems used for more dedicated computation tasks. In embedded microprocessors there are mostly fixed point number system used as a faster and a less power consuming choice – Intel. However, fixed points representation requires to know in advance the range of values which in DNNs is unknown before a training. In addition the ranges may vary accross different network layers that requires an additional data processing to perform computations. It makes this number system less convienient for DNNs (Reuther et al. 2020).

In the last decades as an alternative for fixed point and floating point number systems, a logarithmic number system was investigated (Swartzlander and Alexopoulos 1975). It allows to use a wider range of exactly represented values and makes multiplications and divisions easier than in the floating point number system. However, additions and substractions in logarithmic domain might be a non trivial task. A more detailed description and comparison of these number systems could be find in (Chugh and Parhami 2013; Parhami 2020; Haselman et al. 2005).

2.3.1 IEEE 754 Standard

The Institute of Electrical and Electronics Engineers (IEEE) standarized it in 1985 in IEEE 754 Standard for Floating-Point Arithmetic technical report (P754 1985). Right now it is the most commonly used standard in a modern computing.

The floating point number system F with the base equal to 2, is a finite subset of real values where every element $v \in F$ is in the following form:

$$v = (-1)^s \, 2^{e-t} \, (1 + m) \tag{2.11}$$

where:

- The sign bit s is equal to 0 or 1 for positive and negative values, respectively.
- The integer e stands for an exponent.
- The value m is a fraction (also called a mantissa).
- The exponent bias $t = 2^{\#exponent\_bits-1} - 1$ move the exponent range symmetrically (in increments to 1) around zero to allow obtaining negative exponents. That makes possible to represent values smaller than 1.

When we have 4 bits exponent then $e \in (0, 15)$ takes only non negative values but $e - t \in (-7, 8)$ allows to represent the negative values too. Thus the reciprocal of a floating point is a floating point too. It is also important to mention that in IEEE 754 the floating point values with exponent containing all zeros or ones are reserved for special values like for example infinity (Goldberg 1991).

It is worth noting that floating point numbers are not equally spaced in F (see Equation (2.11)). However, they are uniformly spread in subranges of F. On Figure 2-5 there are denoted floating point numbers for 3 bits mantissa. We have eight possible values of m and they are equally spaced inside each of the range: $(2^{-1}, 2^0)$, $(2^0, 2^1)$ and $(2^1, 2^2)$ etc. So the distance between the closest floating points is equal to $1/16$, $1/8$ and $1/4$ for each of the range, respectively.

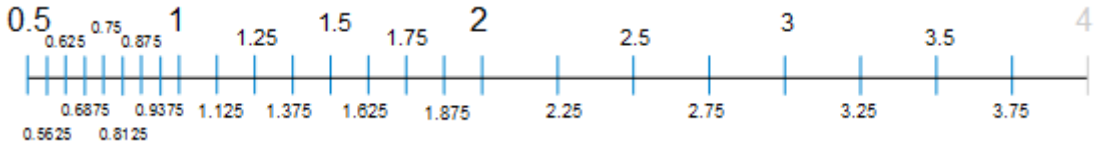


Figure 2-5: Floating point numbers for three bits mantissa (figure taken from (Higham 2002)).

IEEE 754 provides two main formats for a number representation: single (float32/fp32) and double(float64/fp64). The first one uses 32 bits with 8 bits exponent and the second one 64 bits with 11 exponent bits. Thus bias is equal to 127 and 1023 for a single and a double precision, respectively.

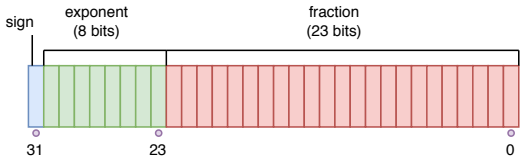


Figure 2-6: Single precision

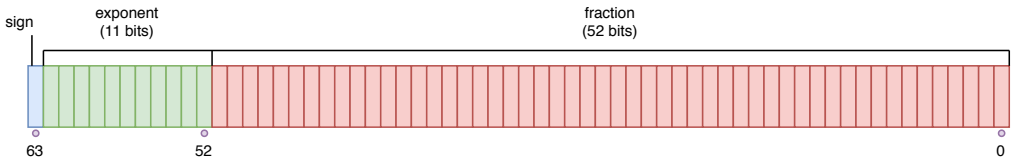


Figure 2-7: Double precision

It is not possible to represent exactly all real numbers in the floating point number system. Firstly, a given number might have to big or to small order of magnitude or its binary representation could be infinite or too long to be represented using available number of bits. In this case the values are truncated or rounded during a conversion which results in an approximated, not the exact representation. The difference between a real value and its floating point representation is called the floating point error. Even a small error then propagates with all operations and might have a significant impact on the final result of computations. In general, the more bits the better (more accurate) representa-

tion. On the other hand storing values in a double precision uses twice as much memory as in a single precision.

2.3.2 Floating points error

In order to perform all kinds of computations we need to represent real values in a limited precision. In this thesis we focus on a floating point representation and the most commonly used IEEE 754 standard we described in Section 2.3. Mapping the infinite field of real values into the finite set of floating points results in a representation error that propagates through all the computations. Also every operation might introduce an additional error. In the case of CNNs due to a high number of convolution operations it could have a huge impact on the accuracy of the final result.

To represent a real value x in a floating point number system it needs to be rounded to the value that can be exactly represented in that system.

Definition 15. *A conversion from the field of real numbers $\mathbb{R}$ to a floating point subset $F \subset \mathbb{R}$ is a mapping function $x \rightarrow fl(x)$ such that $fl(x) = \hat{x}$ where $\hat{x} \in F$ and $\min_{y \in F} |y - x| = |\hat{x} - x|$. It is also called rounding to the nearest operation.*

If the absolute of a real value is larger/smaller than the absolute of the largest/smallest value exactly representable in a given floating point number system, it is referred as an over/under flow appears. In the absence of an over/under flow we define the floating point representation error as an absolute of the difference between the real value and its floating point representation (see Equation (2.13)). As every real value x that is not exactly represented in F and no under/overflow, falls between two consecutive floating points, we can bound the possible maximum floating point representation error by the half of the distance between them. As floating point numbers are not equally spaced, the floating point representation error boundary is not a constant and it depends on which subrange of F the value x falls into (see Figure 2-5). In the presented example values $x \in (1, 2)$ have a maximum floating point error equal to $1/16$ and values $x \in (2, 4)$ have an error boundary equal to $1/8$.

Following Higham's notation (Higham 2002) we rewrite the mapping function as

$$fl(x) = x(1+\delta) \tag{2.12}$$

where $x, \delta \in \mathbb{R}$ and $|\delta| < \varepsilon$ denotes the rounding error of x . The value ε is called a machine epsilon and it is equal to the half of the distance between 1 and the next floating point (see (Goldberg 1991; Demmel 1997)) - Sometimes this value is called a roundoff unit while the machine epsilon is a distance between 1 and the next floating point value. In this thesis we stay with Goldberg and Demmel definition. For a different floating point number systems there are different values of epsilon.

Using Equation (2.12) we define the floating point error as the absolute difference between the real and the mapped values, so the representation error has the following boundary

$$error = |x - fl(x)| = |x - x(1+\delta)| \leq |x|\varepsilon \tag{2.13}$$

Similarly, provided there is no under/over flow the error of the floating point arithmetic operators can be described as follows

$$fl(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta) \tag{2.14}$$

where $|\delta| \leq \varepsilon$, $x, y \in F$ and $\operatorname{op} \in \{+, -, *, /\}$. It is important to notice that we consider the floating point error introduced by performing the operation regardless of the x, y representation error, that why $x, y \in F$ not $x, y \in \mathbb{R}$ (Higham 2002; Goldberg 1991; Wilkinson 1994).

It is important to notice that rounding errors are determined in a deterministic, not a random way. A wide variety of random sampled values with uniform and normal distributions were examined in (Dahlqvist, Salvia, and Constantinides 2019). They came to the conclusion, that rounding errors give generally the same distribution curve, regardless the initial distribution (uniform with $\mu = 0, \sigma = 2$ or normal with $\mu = 2, \sigma = 10$). According to (Kahan 1996)

rounding errors are actually not random, often correlated and behave more like discrete than continuous variable. Additionally very often only a few (two or three) roundings errors are the dominant contributors to the final error. All these facts undermine applicability of Central Limit Theorem to investigate the floating point error behaviour and makes an average case analysis unreliable.

2.3.3 Half precision

Since deep neural networks require to store a lot of data, the trend is to store them in lower precisions and perform fused computations in $float32$. This technique also allows to speed up the data flow and consequently the computational time. Converting values from 32 to 16 bits representation allows to save a memory space but has an impact on the precision of the result, which might degrade the accuracy.

float16

Half precision format represents real values on 16 bits with a one sign bit, 5 exponent bits and 10 mantissa bits (see Figure 2-8). It was added to IEEE 754 standard in 2008 to support a reducing of a memory usage.

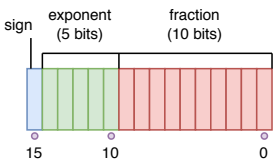


Figure 2-8: Half precision - $float16$

It saves a half of a memory space comparing to a single precision but results not only in a worse accuracy but also in a much smaller range of the representable values. Thus performing computations in $float32$ and storing intermediate results in $float16$ is a problematic idea because of a lot of potential under/over flows (see Table 2.1).

bfloat16

The Brain Floating Point format ($bfloat16$) was first proposed by Google and Intel (Young Cliff 2018; Kloss Carey and Group 2019) and was designed specifically for machine learning algorithms. It occupies 16 bits with a one sign bit, 8 exponent bits and 7 mantissa bits. Thus it is a kind of a truncated single precision format. As the exponent is represented on 8 bits exactly as in $float32$ it allows to keep nearly the same range of representable values as a single precision. In contrast to $float16$ where an exponent is represented on 5 bits $bfloat16$ format allows to avoid a big number of under/over flow cases that appear when casting to the $float16$ precision. The $bfloat16$ number systems is currently supported in a wide range of processors, softwares and frameworks Nervana NNP-T1000 , Intel AVX-512, Google CLoud TPU, TensorFlow, ARM, AMD and CUDA.

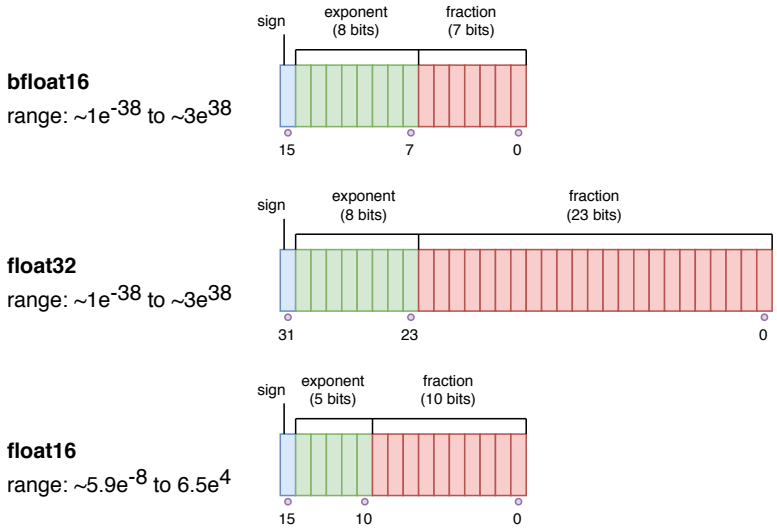


Figure 2-9: $float32$, $float16$ and $bfloat16$ number systems comparison.

In the Table 2.1 we present the important parameters of all described floating point number systems such as machine epsilon minimum and maximum, exactly represented values as well as a number of bits. It provides the trade-offs between the required memory and the accuracy in each case.

	bits	sign	exponent	mantissa	bias	range	ε
float64	64	1	11	52	1023	$10^{\pm 308}$	2^{-53}
float32	32	1	8	23	127	$10^{\pm 38}$	2^{-24}
float16	16	1	5	10	15	$10^{\pm 4}$	2^{-11}
bfloat16	16	1	8	7	127	$10^{\pm 38}$	2^{-8}

Table 2.1: Number of bits dedicated to sign, exponent and mantissa, exponent bias, approximate range and machine epsilon for $float64$, $float32$, $float16$ and $bfloat16$.

2.4 Convolution

In CNNs we use the special case of convolution expressed in Definition 4 that is a discrete convolution or discrete convolution of polynomials. It is a linear operation defined on two polynomials represented by vectors and it is equal to a sum of the products of the corresponding coefficients [2].

Definition 16. *For any two polynomials of a degree n, $w(a), x(a) : \mathbb{R}^n \to \mathbb{R}$ and the corresponding vectors $\mathbf{w}, \mathbf{v} \in \mathbb{R}^n$ the convolution of $\mathbf{w}$ and $\mathbf{x}$ is expressed by the following formula*

$$(\mathbf{w} * \mathbf{x}) = \sum_i w_i x_{n-i} \tag{2.15}$$

That is the convolution of two polynomials of the same degree is equal to the leading coefficient of the polynomial $s(a) = w(a)x(a)$.
Analogously

Definition 17. *For two matrices $\mathbf{W}, \mathbf{X} \in \mathbb{R}^{n \times m}$, corresponding polynomials $W(a, b), X(a, b) : \mathbb{R}^{n \times m} \to \mathbb{R}$ and corresponding vectors $\text{vec}(\mathbf{W}), \text{vec}(\mathbf{X}) \in \mathbb{R}^{nm}$ the convolution is expressed by the following formula*

$$(\mathbf{W} * \mathbf{X}) = \sum w_{ij} x_{(n-i)(m-j)} \tag{2.16}$$

[2]There is an inconsistency in the notation in the recent literature. Formally, the actual mathematical operation is called a cross-correlation (inst.eecs.berkeley. edu and (Tolimieri, An, and Lu 1997)) called convolution in machine learning/engineering field, by a convience. Convolution differs from it by the reversed order of coefficients (elements) in a one of the vector/matrix. We stay with machine learning/engineering field notation

That is the convolution of two polynomials $W(a,b), X(a,b) : \mathbb{R}^{n\times m} \to \mathbb{R}$ is equal to the leading coefficient of the polynomial $S(a,b) = W(a,b)X(a,b)$.

Multiple channels

In convolutional layers, there are multiple input and output channels (see Section 2.1.2). For every input and output channel of the data we have a different set of weights (kernels). The single output channel result is produced by performing convolution on every input channel separately and then the obtained corresponding results are added.

A one-dimensional convolution over C input channels of two sets of vectors of the size n - input $\mathbf{x_c} = [x_{c1}, \ldots, x_{cn}]$ and weights $\mathbf{w_c} = [w_{c1}, \ldots, w_{cn}]$ for all input channels $c = 1, \ldots, C$ - is a scalar defined as follows

$$(\mathbf{w} * \mathbf{x}) = \sum_c \sum_i w_{ci}\, x_{ci} = \sum_{c,i} w_{ci}\, x_{ci} \tag{2.17}$$

A two-dimensional convolution over C input channels of two sets of matrices of the size $n \times n$ - input $\mathbf{X_c} = [x_{cij}]$ and weights $\mathbf{W_c} = [w_{cij}]$ for all input channels $c = 1, \ldots, C$ and $i, j = 1, \ldots, n$ is a scalar defined as follows

$$(\mathbf{W} * \mathbf{X}) = \sum_c \sum_{i,j} w_{cij}\, x_{cij} = \sum_{c,i,j} w_{cij}\, x_{cij} \tag{2.18}$$

2.4.1 Direct/Naive convolution algorithm

A direct convolution algorithm (see Algorithms 1 and 2) computes the single output value result (scalar) of a one or two-dimensional convolution in the straightforward way. That means that we sum up the products of the corresponding values of an input and a kernel. With this approach we have to perform n multiplications and $n-1$ addition to obtain the single output value result in a one-dimensional case and n^2 multiplications and $n^2 - 1$ additions in a two-dimensional case.

ALGORITHM 1: One-dimensional, single output value direct convolution algorithm

Input: n - size of the input, ;

$\mathbf{w} = [w_1, \ldots, w_n]$ - weights/kernel,;

$\mathbf{x} = [x_1, \ldots, x_n]$ - input

Output: s - convolution result

$s = 0$;

for $i = 1$ *to* n **do**
    $s+ = w_i \, x_i$

end

ALGORITHM 2: Two-dimensional, single output value direct convolution algorithm

Input: $n \times n$ - size of the input ;

$\mathbf{W} = [w_{ij}]$ for $i, j = 1, \ldots, n$ - weights/kernel ;

$\mathbf{X} = [x_{ij}]$ for $i, j = 1, \ldots, n$- input;

Output: s - convolution result

$s = 0$;

for $i = 1$ *to* n **do**

    for $j = 1$ *to* n **do**
        $s+ = w_{ij} \, x_{ij}$

    end

end

Thus, to compute a single output value using the direct algorithm we need $O(n)$ operations for a one-dimensional kernel and an input and $O(n^2)$ for a two-dimensional case. Using a direct convolution algorithm (see Algorithm 1 and Algorithm 2) we compute every output value separately, so the complexity will be equal to the input size ($width \times height \times depth$) multiplied by the complexity of single output value computations, namely $O(n)$ or $O(n^2)$ for a one and two dimensions respectively. It means that a huge amount of time is spent on convolution computations in CNNs. Hence it is a good reason to look for faster options.

2.5 Fast Convolution Algorithms

An algorithm, from the general point of view, is a sequence of steps that indicate how to compute an output from a given input. However, very often there is more than a one way to solve the problem and thus more than one algorithm. The term fast convolution algorithm is understood as the algorithm that computes an output in a faster way (complexity regarding a number of performed operations) than the classic ones. This effect is obtained mainly by reducing the number of multiplications. For example we can compute $ac + ad + bc + bd$ as $(a+b)(c+d)$ (Blahut 2010). The first method requires 4 multiplications and three additions, while the second one only one multiplication and two additions.

Most of the fast convolution algorithms consist of three steps: linear transformations of both inputs, namely input and weights/kernel, element-wise multiplication of them (Hadamard product) and linear transformation of the obtained result.

One of the core differences between the direct convolution algorithm and algorithms presented below is that using the direct algorithm (see Algorithm 1 and Algorithm 2) we compute only a one output value (scalar) at a time. To compute vector of the size m output values we need to compute each vector element (output value) separately that means to perform m times the convolution computations described in the algorithm (see Algorithms 1 and 2, for one and two-dimensional convolution respectively). In fast convolution algorithms, we compute several output values (output vector) taking the bigger input tile. This approach results in a lower number of overlapping values (see Section 2.1.2) than in a direct algorithm Algorithms 1 and 2. It results in a fewer number of multiplication operations and thus in reduction of time complexity. We will discuss this problem in more details in Chapter 5. An interesting overview of the fast convolution algorithms can be found in (Blahut 2010; Ju and Solomonik 2020).

It is important to be aware that despite there being two algorithms solving the same problem and theoretically compute identical output for the same input, in practice the results might be different. The reason of this fact lies in a

finite number representation (see Section 2.3). Different kinds and order of operations could result in a different floating point error arising from the limited precision representation. In this section we briefly describe most popular fast convolution algorithms and highlight their disadvantages that are removed by using Winograd algorithm.

2.5.1 DFT convolution

One of the most popular fast convolution algorithm uses Discrete Fourier Transform (DFT) to perform linear transformations. The idea of Discrete Fourier Transform (DFT) convolution is to transform an input and a kernel into an other domain, where the convolution becomes an element-wise multiplication (Hadamard product) and later to transform the obtained result back (see Figure 2-10). To keep the numerical error of transformation computations small even for big input/kernel sizes the transformation matrices are constructed in the complex number field (Higham 2002). For a fixed kernel size k to compute $m = n - k + 1$ output values for a one-dimensional convolution every transformation requires $O(n^2)$ complex multiplication operations and to compute element-wise multiplication in the field of complex numbers we have to perform $O(n)$ complex multiplications (see Figure 2-10). Each of the complex multiplication consists of at least three real multiplications (Karatsuba and Ofman 1962). It makes DFT convolution algorithm not practical in the context of the number of multiplication operations, and in the resulted time complexity.

2.5.2 FFT convolution

The wide application of Fourier algorithm is due to the existence of the Fast Fourier Transform algorithm (FFT). The FFT permits rapid computation of the DFT by decreasing the complexity of transformation operations. Namely, FFT decreases the complexity of whole convolution computations from $O(n^2)$ to $O(n \log n)$ (see Figure 2-11). The reduction of the number of multiplications in convolution computations is significant especially for a large data size n.

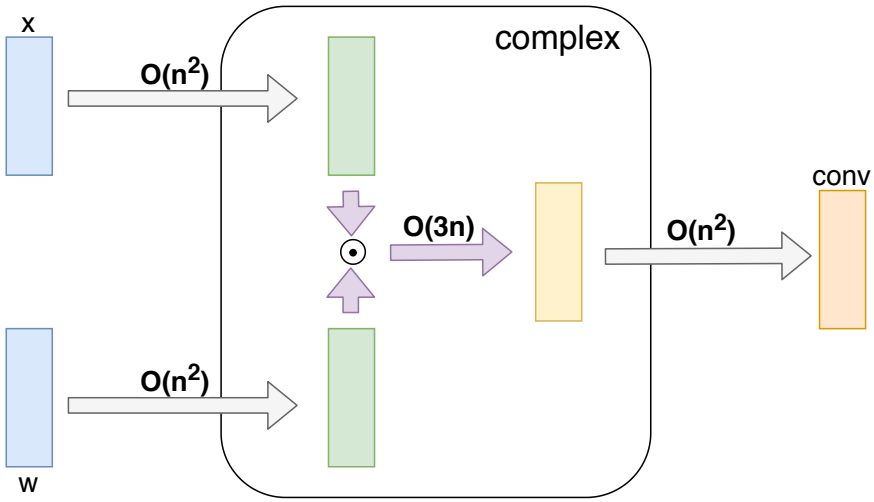


Figure 2-10: DFT convolution algorithm.

There are several different implementations of FFT. The first algorithm was presented by Good (Good 1958; Good 1960) and Thomas (Thomas 1963). The most commonly used variant of FFT is from Cooley and Tukey (Cooley and Tukey 1965). And the most efficient, but also the most complicated one was proposed by Winograd (Winograd 1976; Winograd 1978). The detailed analysis of various versions of FFT algorithms can be found in (Nussbaumer 1981).

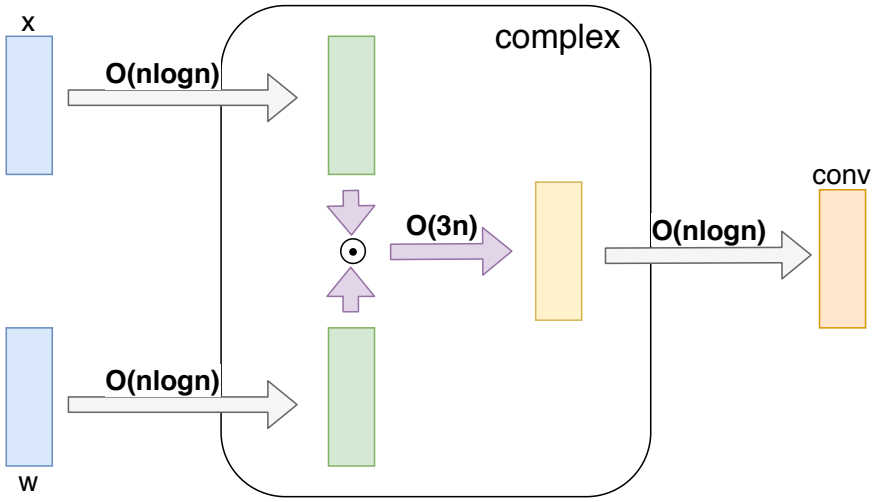


Figure 2-11: FFT convolution algorithm

Applying FFT reduces the complexity of transformations compared to DFT in particular for bigger output/kernel sizes but the core convolution still con-

sists of n complex multiplications. That means at least $3n$ real multiplications have to be executed. In CNNs we use the same input and kernel several times (see Section 2.1.2), so the transformation complexity is amortized over multiple uses. The number of real element-wise multiplications has more significant impact on the time complexity. In FFT convolution these multiplications are executed in the field of complex numbers and we have to perform at least $3n$ real number multiplications to compute the product of two complex values. If we perform the core convolution operations in the field of real numbers then it reduces the number of them from $3n$ to n.

2.5.3 DTT convolution

An interesting alternative for DFT and FFT convolution algorithms, where Hadamard product is computed in the field of complex numbers might be a discrete trigonometric transform (DTT) where Hadamard product is performed in the field of real numbers. Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST) are alternative methods of linear transformations used in DFT and FFT convolution algorithm. The transformation might be performed in the same way as FFT reducing whole computational time from $O(n^2)$ to $O(n \log n)$. However, transformation matrix elements are irrational and this introduces a representation error that propagates further. As we show in Chapter 5 for small kernel sizes used in CNNs this propagation is very important for the final accuracy. Very popular Chebyshev points widely used to construct Vandermonde matrices in numerical computations works good for bigger matrix sizes (greater than 10) (Trefethen and Bau 1997). Moreover, their main benefit is that using Chebyshev points mitigate Runge effect that is important for continuous not discrete computations. Similarly Chebyshev points are mostly irrational for bigger transformation matrix sizes.

2.5.4 Winograd/Toom-Cook convolution

The family of algorithms commonly known in the machine learning field as the Winograd convolution was firstly presented by Toom (Toom 1963) and Cook (Cook 1966). Their work is a generalization of the Karatsuba method (Karatsuba and Ofman 1962) (translation in (Karatsuba and Ofman 1963)) (Karatsuba 1995) implemented to decrease the complexity of a polynomial multiplication. Shmuel Winograd applied the Toom-Cook algorithm to convolution computations in a signal processing field in 1980s (Winograd 1980b). He also has done the significant theoretical work on this algorithm. He proved that for fixed output and kernel sizes Toom-Cook algorithm performs convolution in the minimum possible number of real element-wise multiplication operations which Winograd refers to as a general multiplication (Winograd 1980a).

The formal description of the Toom-Cook convolution algorithm can be found in (Blahut 2010; Tolimieri, An, and Lu 1997). However, for those who want to implement Winograd/Toom-Cook convolution it can be challenging to extract algorithm from the theory and examples.

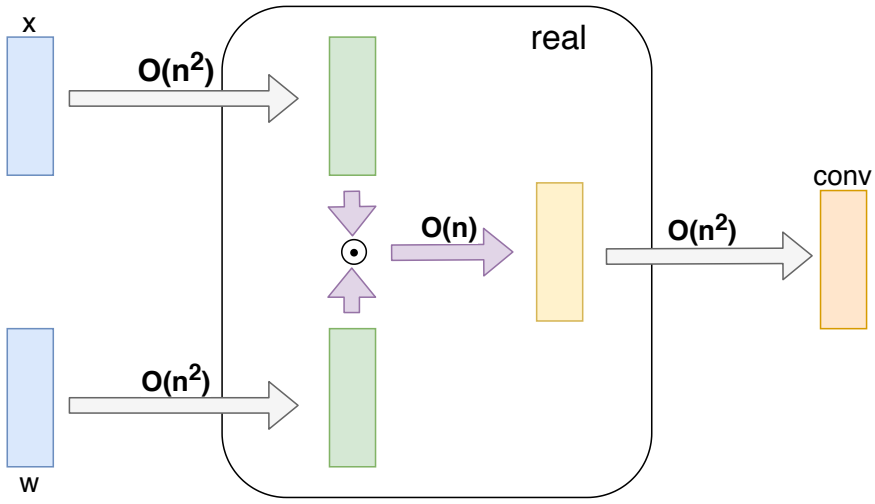


Figure 2-12: Winograd convolution algorithm.

The Winograd/Toom-Cook convolution algorithm is based on the same idea as the DFT convolution algorithm but an element-wise multiplication is done in the field of real numbers instead of the complex one (see Figure 2-12). Thus, as was mentioned above, every product of two real values requires

only one multiplication in contrast to the complex field where it needs at least three multiplications to compute product of any two complex values (see Algorithm 3).

Moving into the field of real numbers and decreasing the number of general multiplications is done at the cost of pre/post processing operations that have $O(n^2)$ complexity. In convolutional neural networks however, it is not a big inconvenience for two reasons. Firstly, because the kernel sizes are typically small and secondly, because we use the same input and kernels several times, so the preprocessing operations are amortized over multiple uses. Post processing operations are done after summation over input channels (Lavin and Gray 2016) which also reduces the time (see Algorithm 4). The serious disadvantage is that these pre and post-processing operations have bad properties regarding the computational error. Because of this, in contrast to DFT and FFT convolution computations, we can get inaccurate results particularly for bigger input/kernel sizes. DTT transformation are known for the best properties for Vandermonde matrices in the field of real numbers. However matrix elements are irrational that generate additional representation error. For such small sizes of matrices as used in CNNs the DTT convolution computation result has greater numerical error than the result obtained with Toom-Cook/Winograd algorithm.

Lavin and Gray (Lavin and Gray 2016;) wrote the seminal paper on applying Winograd/Toom-Cook convolution to deep neural networks. They showed how to apply two-dimensional $F(2 \times 2, 3 \times 3)$ and $F(4 \times 4, 3 \times s)$ algorithms to DNN convolution with multiple input and output channels. They also used distributive property to amortize the cost of post-processing operations over multiple convolutions. They tested proposed algorithms on Vgg network in $fp32$ and $fp16$. Comparing to Nvidia's DNNs library cuDNN Winograd/Toom-Cook gave up to $2.64\times$ speed up in convolution computations. However they noticed that the error of computations grows significantly with output size. In $fp32$ the floating point error of Toom-Cook/Winograd convolution computa-

ALGORITHM 3: Two-dimensional, single channel, multiple output point Winograd convolution algorithm

Input: $\mathbf{W} = [w_{ij}]$ for $i, j = 1, \ldots, k$ - weights/kernel ;

$\mathbf{X} = [x_{ij}]$ for $i, j = 1, \ldots, n$- input;

Output: $\mathbf{S} = [s_{ij}]$ for $i, j = 1, \ldots, n - k + 1$ - convolution result

$\mathbf{W^{(Win)}} = Preprocess(\mathbf{W})$;

$\mathbf{X^{(Win)}} = Preprocess(\mathbf{X})$;

for $i = 1$ *to* n **do**

  for $j = 1$ *to* n **do**

    $s_{ij}^{(Win)} = w_{ij}^{(Win)}\, x_{ij}^{(Win)}$

  end

end

$\mathbf{S} = Postprocess(\mathbf{S^{(Win)}})$

ALGORITHM 4: Two-dimensional, multiple channels, multiple output value Winograd convolution algorithm

Input: $\mathbf{W} = [w_{cij}]$ for $i, j = 1, \ldots, k$, $c = 1, \ldots, C$ - weights/kernel for C channels;

$\mathbf{X} = [x_{cij}]$ for $i, j = 1, \ldots, n$, $c = 1, \ldots, C$ - input for C channels;

Output: $\mathbf{S} = [s_{ij}]$ for $i, j = 1, \ldots, n - k + 1$ - convolution result

for $c = 1$ *to* C **do**

  $\mathbf{W_c^{(Win)}} = Preprocess(\mathbf{W_c})$;

  $\mathbf{X_c^{(Win)}} = Preprocess(\mathbf{X_c})$;

end

for $c = 1$ *to* C **do**

  for $i = 1$ *to* n **do**

    for $j = 1$ *to* n **do**

      $s_{ij}^{(Win)} = s_{ij}^{(Win)} + w_{cij}^{(Win)}\, x_{cij}^{(Win)}$

    end

  end

end

$\mathbf{S} = Postprocess(\mathbf{S^{(Win)}})$

tions for $F(2 \times 2, 3 \times 3)$ was $10\times$ smaller than the error of direct convolution computations. For $F(4 \times 4, 3 \times 3)$ the error was $10\times$ greater than error of direct

convolution. In $fp16$ all algorithms gave $100\times$ greater error than direct convolution in $fp32$. Although they used the Toom-Cook algorithm to generate their core convolution algorithms, they referred to it as the Winograd convolution. This has become the accepted term in the DNN literature. They presented transformation matrices in only two cases for the output of the size 2×2 and 4×4 and the kernel of the size 3×3 but they do not provide any general way of how to construct transformation matrices for other parameters (root points) and other output/kernel sizes.

Some research on the optimality of the Toom-Cook algorithm was done by M. Bodrato (Bodrato 2007; Bodrato and Zanoni 2007; Bodrato and Zanoni 2006) He focused on the optimality of this algorithm applied to the polynomial multiplication problem, as measured by the number of the required operations. Improving the numerical accuracy of the result was not a goal of Bodrato's work, and no data is provided to the effect of the proposed techniques on numerical accuracy.

So far there is very few results on the floating point accuracy of the Winograd /Toom-Cook convolution algorithm. Vincent et al. (Vincent et al. 2017) proposed to scale transformation matrices to improve their properties (Gautschi 2011). They demonstrated that this approach can reduce the error in exactly one case: convolving a 5×5 kernel to create a 9×9 output block. Further they showed, that this improved matrices could be successfully used for training a DNN. However, they did not provide a method for choosing good scaling factors.

At the same time as a research presented in this thesis were done, several papers which treat about Toom-Cook/Winograd algorithm were published.

In (Zlateski et al. 2018) authors demonstrate that on existing architectures the Toom-Cook/Winograd convolution algorithms does not achieve their theoretical time complexity. A lot of time is spent on data flow. They also proposed a couple of optimization techniques to speed up Toom-Cook/Winograd convolution computations.

A couple of papers focus on efficient Toom-Cook/Winograd algorithm implementation using various compression methods. William Dally's group investigated pruning method (Han et al. 2015). They replace some values in a transformed input and transformed weights with zeros, making matrices sparse. That allows to perform convolution faster because decreasing number of non-trivial multiplications (Liu et al. 2018). For algorithm $F(2 \times 2, 3 \times 3)$ they reduced the number of multiplication operations between $6.8\times$ and $10.8\times$ dependent on dataset without significant accuracy loss (less than 0.1%)

An interesting idea that uses kernel decomposition was presented in (Maji and Mullins 2018). The matrix of weights is split into two vectors whose outer product is an approximation of the weight matrix. It reduces the space required to store weights and consequently reduces the data flow that speeds up convolution computations up to $8\times$ (tested on Vgg network) comparing to existing implementations. Reducing $2D$ convolution into $1D$ by this decomposition also reduce the number of multiplications. However, an additional work on kernel decomposition and layers reconstruction is required.

Some other research were done on using lower precision representations. Special attention was given to unsigned 8 bits integer number system. The other solution was to use Winograd algorithm with three linear and one superlinear polynomial that gave mix of real and complex numbers (Meng and Brothers 2019). They tested their solution on various networks using convolution $F(6 \times 6, 3 \times 3)$. To improve data flow they convert weights in Winograd domain to $int8$. They had $3\times$ complexity reduction comparing to a direct algorithm without any significant loss of accuracy.

A lot of focus was on speed up Toom-Cook/Winograd convolution computations tuning implementation to particular architectures. One of the paper presents efficient Toom-Cook/Winograd algorithm focus on performance on Arm processors (Maji et al. 2019). They propose the optimization method that uses General Matrix Multiplication to perform Hadamard product computations over multiple channels for each input tile. As there exists a lot of efficients

GEMM implementations the speed of the convolution computations increases $4\times$ in comparision to standard Arm convolution implementation (Im2row).

Later (Fernandez-Marques et al. 2020) investigated training of neural networks using Toom-Cook/Winograd convolution algorithm with values represented in $int8$. They treated transformation matrices as trainable parameters to improve the accuracy of the convolution computation results. They get $2.7\times$ speed up comparing to the direct/naive algorithm (Arm - Im2row implementation) with marginal accuracy drop comparing to Toom-Cook/Winograd computations in $fp32$.

There is also Nvidia work presented in (Liu, Yang, and Lai 2021) where Tensor cores are used for Toom-Cook/Winograd algorithm implementation in mixed precisions. It gives $2.41\times$ speed up comparing to GEMM based cuDNN Toom-Cook/Winograd implementation without the accuracy loss (tested on Vgg).

Some published research focuses on Toom-Cook/Winograd convolution algorithms in other than floating points and $int8$ number systems. However this approach always requires some additional costs somewhere else. For this reason they are not so popular in machine learning computations as floating point number system (see Section 2.3).

The residue number system (RNS) based on modulo arythmetic was proposed by (Liu and Mattina 2020). They tested inference in 8 bits values representation using pretrained quantized networks. For Vgg network they got over $2\times$ better time performance without drop in accuracy for output size up to 16×16 comparing to $int8$ for Arm architectures. However, the conversion into and from RNS is time consuming, particularly for longer tiles.

Logarithmic number system was investigated in (Alam and Gregg 2020; Alam, Garland, and Gregg 2021). They found that most popular base 2 is not the best choice for performing Toom-Cook/Winograd convolution computations in lower precisions. Using base 1.851 instead it is possible to reduce the error of conversion from 8 bits floats (minifloats) $10\times$ compared to base 2.

2.5.5 Winograd convolution

Winograd developed a separate family of convolution algorithms mixing real and complex numbers in the element-wise multiplication stage (Winograd 1980a). His method creates a much larger set of algorithms than Toom-Cook's, including algorithms that are not optimal with respect to the number of general multiplications. However, because the number of overlapings is smaller for bigger input sizes, by using them we can still obtain the optimal number of multiplications for the whole input. This observation will be discussed in more details in Chapter 6.

The mathematical background and example of matrices for a 'nearly optimal' algorithm are described by Blahut (Blahut 2010), Parhi (Parhi 2007) and Tolimieri (Tolimieri, An, and Lu 1997). Selesnick and Burrus (Selesnick and Burrus 1994) investigated Winograd convolution algorithms in greater details. However, their focus was on the number of general multiplications, not on the accuracy of computations. Meng and Brothers in (Meng and Brothers 2019) successfully applied one version of Winograd algorithm in DNNs to perform computations in lower precisions.

Chapter 3

Toom-Cook Convolution Algorithm

The Toom-Cook convolution algorithm is proved to be the most optimal one in terms of the number of general multiplications. It was presented firstly around fifty years ago by Toom (Toom 1963) and Cook (Cook 1966). Around thirty years ago Winograd applied this method in signal processing computations. In 2016 Lavin and Gray (Lavin and Gray 2016) used the simplest/smallest version to the DNNs convolution computations with the kernel of the size 3×3 to compute 2×2 output values at a time.

In this chapter we present the theoretical background provided by Schmuel Winograd (Winograd 1980b). The novelty is to provide an exact algorithm for computing the elements of transformation matrices. It allows us to construct a multiple versions of the algorithm for given kernel and input/output sizes.

3.1 Toom-Cook algorithm definition

The Toom-Cook algorithm is based on the Chinese Remainder Theorem (CRT) for polynomials (Theorem 2) and the Matrix Exchange Theorem (Theorem 3). The main idea of this algorithm is to transform the kernel and the input into the other domain in machine learning field called 'Winograd domain' where convolution becomes an element-wise multiplication (Hadamard product Definition 1) and then transform the result back. To perform convolution computations we need to construct three matrices, first one to transform the kernel,

second one to transform the input tiles and third one to transform the results back.

Theorem 2 (Chinese Remainder Theorem for polynomials)**.** *Let $\mathbb{F}[a]$ denote a ring of all polynomials over a field $\mathbb{F}$. Let $s_1(a), \ldots, s_\ell(a) \in \mathbb{F}[a]$ be arbitrary polynomials and $m_1(a), \ldots, m_\ell(a) \in \mathbb{F}[a]$ denote irreducible polynomials such that $GCD(m_i(a), m_j(a)) = 1 \; \forall i, j = 1, 2, \ldots, \ell, \; i \neq j$.*
Then the system of congruences

$$s(a) = s_i(a) \mod m_i(a) \qquad \textit{for} \;\; i = 1, \ldots, \ell$$

has exactly one solution $s(a)$ modulo $M(a) = m_1(a) \ldots m_\ell(a)$ given by the formula

$$s(a) = \sum_{i=1}^{\ell} s_i(a) N_i(a) M_i(a) \mod M(a)$$

where for $i = 1, 2, \ldots, \ell$:

$$N_i(a) M_i(a) = 1 \mod m_i, \qquad \textit{and} \qquad M_i(a) = \frac{M(a)}{m_i(a)}$$

In what follows we assume that $\mathbb{F}$ is a field of real numbers $\mathbb{R}$ and that a kernel vector $\mathbf{w} = [w_1, w_2, \ldots, w_k]$ and a input vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ are represented as polynomials $w(a)$ and $x(a)$, with coefficients equal to their respective components, such that the leading coefficients of $w(a)$ and $x(a)$ are w_k and x_n, respectively. Namely $w(a) = w_1 + w_2 a + \ldots + w_k a^{k-1}$, $x(a) = x_1 + x_2 a + \ldots + x_n a^{n-1}$ (see Section 2.2.1). Then computing the one-dimensional discrete convolution is equivalent to computing the coefficients of the polynomial product $s(a) = w(a)x(a)$.

In the Toom-Cook algorithms, it is assumed that all $m_i(a)$ are monomials, so the computation reduces to the following steps:

1. Choose ℓ pairwise different root points p_i to construct polynomials $m_i(a) = a - p_i$ for $i = 1, \ldots, \ell$.

2. Evaluate polynomials $w(a)$, $x(a)$ at each root point p_i to change the domain, which is equivalent to computing $w_i(a) = w(a) \mod m_i(a)$ and $x_i(a) = x(a) \mod m_i(a)$ for $i = 1, \ldots, \ell$.

3. Perform the multiplication $s_i(a) = w_i(a)x_i(a)$ for $i = 1, \ldots, \ell$.

4. Compute the coefficients of polynomial s_i for $i = 1, \ldots, \ell$ using the Chinese Remainder Theorem.

We can represent the result of this algorithm as:

$$\mathbf{V}^{-1}(\mathbf{V_x x} \odot \mathbf{V_w w}) \tag{3.1}$$

where matrices $\mathbf{V_x}$ and $\mathbf{V_w}$ (referred to as Vandermonde matrices - see Section 2.2.1) represent evaluation of the polynomials $x(a)$ and $w(a)$ in root points p_i for $i = 1, \ldots, \ell$. The matrix $\mathbf{V}^{-1}$ is the inverse Vandermonde matrix used to transform the result back from the 'Winograd domain'. The non-singularity of these matrices is guaranteed by choosing the pairwise distinct root points p_i so that the assumptions of the CRT are fulfilled.

As for monomials $m_i(a) = a - p_i$, it is true that $w(a) \mod m_i(a) = w(p_i)$ this method is equivalent to the following steps:

- Evaluate polynomials $w(a)$ and $x(a)$ in ℓ pairwise distinct root points p_i that are the roots of the polynomials $m_i(a)$. The result is ℓ pairs of values: $w(p_i)$ and $x(p_i)$.

- Mutiply corresponding values $w(p_i)x(p_i)$. Then when define polynomial $s(a)$ as a product of $w(a)$ and $x(a)$ ($s(a) = w(a)x(a)$) we have a values of $s(p_i)$ in ℓ pairwise distinct root points p_i.

- Apply the Lagrange interpolation formula to get the coefficient of the polynomial $s(a)$.

When the kernel is represented as a polynomial $w(a)$ of the degree $k-1$ and input as a polynomial $x(a)$ of the degree $n-1$ then $s(a)$ is of the degree $n+k-2$. To be able to interpolate it $n + k - 1$ pairwise distinct root points are required.

Then the method presented above allows to compute $n - k + 1$ convolution output values in $\ell = n + k - 1$ general multiplications. We can reduce it even further by applying Matrix Exchange Theorem (Blahut 2010):

Theorem 3. *Let* $\mathbf{D}$ *be a diagonal matrix. If the matrix* $\mathbf{M}$ *can be factorised as* $\mathbf{M} = \mathbf{CDE}$ *then it also can be factorised as* $\mathbf{M} = (\overline{\mathbf{E}})^T\mathbf{D}(\underline{\mathbf{C}})^T$*, where matrix* $\overline{\mathbf{E}}$ *is a matrix obtained from* $\mathbf{E}$ *by reversing the order of its columns and* $\underline{\mathbf{C}}$ *is a matrix obtained from* $\mathbf{C}$ *by reversing the order of its rows.*

We can rewrite the Equation (3.1) as $\mathbf{V}^{-1}\mathbf{D_w}\mathbf{V_x}\mathbf{x}$, where $\mathbf{D_w}$ is the diagonal matrix constructed from elements of $\mathbf{V_w}\mathbf{w}$. Applying Matrix Exchange Theorem the formula is equivalent to $\mathbf{V_x}^T\mathbf{D_w}\mathbf{V}^{-T}\mathbf{x} = \mathbf{V_x}^T(\mathbf{V_w}\mathbf{w} \odot \underline{\mathbf{V}}^{-T}\mathbf{x})$.

Although the literature on DNNs typically calls this operation convolution, from the mathematical point of view the operation we want to compute is, in fact, the cross-correlation (see Section 2.4). This is why, when applying the Matrix Exchange Theorem, we do not reverse the order of columns in matrix $\mathbf{E}$. Applying this theorem allows us to reduce the number of multiplications and compute $n + k - 1$ convolution output values in n general multiplication. Putting $\mathbf{A} = \mathbf{V_x}$, $\mathbf{G} = \mathbf{V_w}$ and $\mathbf{B} = \mathbf{V}^{-1}$ we obtain the following formula for one-dimensional convolution (Blahut 2010; Winograd 1980b)

$$(\mathbf{w} * \mathbf{x})_{1D} = \mathbf{A}^T(\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}) \tag{3.2}$$

If we perform convolution computation in DNNs over multiple input channels C we can take an advantage from linear properties of matrix-vector multiplication and compute the result in the following way (Lavin and Gray 2016)

$$(\mathbf{w} * \mathbf{x})_{1D} = \mathbf{A}^T(\sum_c(\mathbf{G}\mathbf{w_c} \odot \mathbf{B}^T\mathbf{x_c})) \tag{3.3}$$

where $\mathbf{w_c}$ and $\mathbf{x_c}$ are the vectors representing weights and input tile for an input channel c.

For two-dimensional convolution we treat the input matrix $\mathbf{X}$ of the size $n \times n$ and weights/kernel matrix $\mathbf{W}$ of the size $k \times k$ as polynomials of two variables: $X(a, b) = \sum_i \sum_j X_{ij}a^ib^j$ and $W(a, b) = \sum_i \sum_j W_{ij}a^ib^j$ (see Section 2.2.1).

The convolution of $\mathbf{X}$ and $\mathbf{W}$ is then equal to matrix $\mathbf{S}$ consisted of the coefficients of the polynomial $S(a,b) = X(a,b)W(a,b)$. To compute coefficients of the polynomia Sl equal to the product of polynomials X and W instead of matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{G}$ we use their Kronecker product $\mathbf{A} \otimes \mathbf{A}$, $\mathbf{B} \otimes \mathbf{B}$ and $\mathbf{G} \otimes \mathbf{G}$ (see Definition 2). We apply it to the vectorised matrices ($\mathrm{vec}(\cdot)$) by stacking their columns into a one column vector (Blahut 2010).

$$(\mathbf{W} * \mathbf{X})_{2D} = (\mathbf{A}^T \otimes \mathbf{A}^T)((\mathbf{G} \otimes \mathbf{G})\,\mathrm{vec}(\mathbf{W}) \odot (\mathbf{B}^T \otimes \mathbf{B}^T)\,\mathrm{vec}(\mathbf{X})) \tag{3.4}$$

Using the Kronecker product properties ($(\mathbf{M} \otimes \mathbf{M})\,\mathrm{vec}(\mathbf{N}) = \mathrm{vec}(\mathbf{M}\mathbf{N}\mathbf{M}^T)$) we obtain a formula for the two-dimensional convolution is as follows

$$(\mathbf{W} * \mathbf{X})_{2D} = \mathbf{A}^T(\mathbf{G}\mathbf{W}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}\mathbf{B})\mathbf{A} \tag{3.5}$$

where matrices $\mathbf{W}$ and $\mathbf{X}$ are the two-dimensional weights/kernel and input, respectively (see Figure 3-1). Similarly as for a one-dimensional convolution formula for multiple channels two-dimensional computations

$$(\mathbf{W} * \mathbf{X})_{2D} = \mathbf{A}^T(\sum_c \mathbf{G}\mathbf{W}_\mathbf{c}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}_\mathbf{c}\mathbf{B})\mathbf{A} \tag{3.6}$$

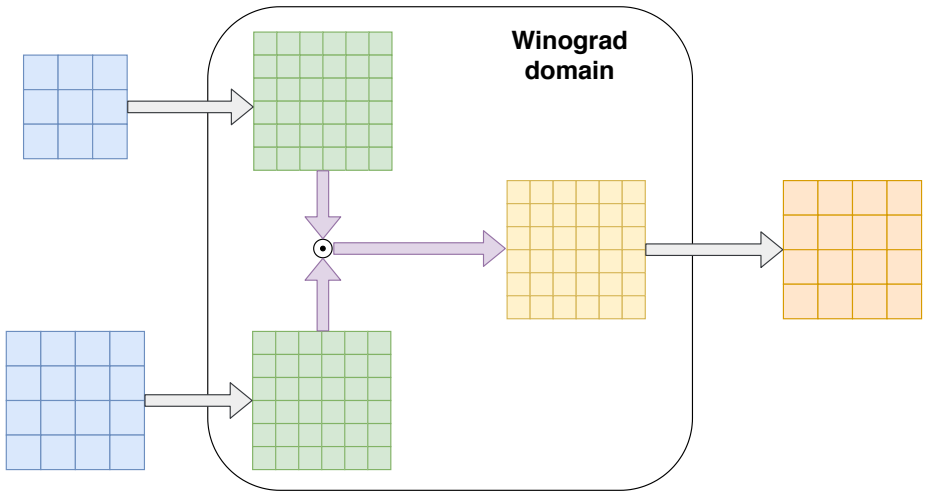


Figure 3-1: Two-dimensional Toom-Cook convolution

3.1.1 Matrices Construction

A method of constructing matrices $\mathbf{A}^T$, $\mathbf{G}$ and $\mathbf{B}^T$ is presented in Algorithm 5. To compute a one-dimensional convolution of the size m with the kernel of the size k, we need an input of the size $n = k + m - 1$. As inputs to the algorithm we provide n pairwise different real root points $p_1, \ldots, p_n$ and use them to construct n linear polynomials $m_i(a) = x - p_i$, for $i = 1, \ldots, n$. We compute polynomial

$$M(a) = (a - p_1) \ldots (a - p_n)$$

and polynomials

$$M_i(a) = M(a)/m_i(a) = \sum_j M_{j,i} a^{j-1} \quad \text{for} \quad i, j = 1, 2, \ldots, n$$

used in Theorem 2. A matrix $\mathbf{A}^T$ is a transposed rectangular Vandermonde matrix of the size $m \times n$. We compute its elements as the zeroth to $(m-1)$th powers of the n selected root points. Next, we construct the matrix $\mathbf{G}$ of size $n \times k$ in a very similar way.

Note that we scale one of the Vandermonde matrices by coefficients N_i to obtain matrices $\mathbf{G}$ and $\mathbf{B}^T$. We find the coefficients N_i using the Euclidean algorithm (Biggs 2002).

The general forms of matrices obtained by the Toom-Cook algorithm is as follows

$$\mathbf{G} = \begin{bmatrix} N_1 & p_1 * N_1 & \cdots & {p_1}^{k-1} * N_1 \\ N_2 & p_2 * N_2 & \cdots & {p_2}^{k-1} * N_2 \\ \vdots & \vdots & \ddots & \vdots \\ N_n & p_n * N_n & \cdots & {p_n}^{k-1} * N_n \end{bmatrix}$$

$$\mathbf{A}^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ p_1 & p_2 & \cdots & p_n \\ \vdots & \vdots & \ddots & \vdots \\ p_1^{m-1} & p_2^{m-1} & \cdots & {p_n}^{m-1} \end{bmatrix} \qquad \mathbf{B}^T = \begin{bmatrix} M_{1,1} & \cdots & M_{1,n} \\ \vdots & \ddots & \vdots \\ M_{n,1} & \cdots & M_{n,n} \end{bmatrix}$$

ALGORITHM 5: Algorithm to construct transformation matrices for Toom-Cook convolution algorithm

Input: m - size of output,

k - size of kernel,

$\{p_1, \ldots, p_n\}$ set of n different root points

Output: Three matrices $\mathbf{A}^T$, $\mathbf{G}$ and $\mathbf{B}^T$ for Toom-Cook convolution

$n = m + k - 1$;

for $i = 1$ *to* n **do**

  for $j = 1$ *to* n **do**

    $M_{i,j}$ = coefficient of the polynomial $\prod_{l \neq i}(a - p_l)$ stands for a^{j-1}

  end

end

for $i = 1$ *to* n **do**

  $N_i = \frac{1}{\prod_{j \neq i}(p_i - p_j)}$

end

for $i = 1$ *to* m **do**

  for $j = 1$ *to* n **do**

    $A^T_{i,j} = {p_j}^{i-1}$

  end

end

for $i = 1$ *to* n **do**

  for $j = 1$ *to* k **do**

    $G_{i,j} = {p_i}^{j-1} * N_i$

  end

end

for $i = 1$ *to* n **do**

  for $j = 1$ *to* n **do**

    $B^T_{i,j} = M_{j,i}$

  end

end

3.2 Modified Toom-Cook algorithm

A common method to reduce the number of terms in the linear transformations of Toom-Cook convolution is to use the so-called modified Toom-Cook algorithm [1].

The main idea of the modified algorithm is to solve a one size smaller problem than in a basic version of Toom-Cook algorithm, what means that we use a kernel of the same size k but an input of the size $n - 1$ instead of n. Having

[1] See tensorflow source code at https://github.com/tensorflow/tensorflow/blob/9590c4c32dd4346ea5c356733 36f5912c6072bf2/tensorflow/core/kernels/winograd_transform.h#L179-L186 and MKL-DNN https://github.com/intel/mkl-dnn/blob/fa5f6313d6b65e8f6444c6900432fb07ef5661e5/doc/winograd_convolution.md

computed such a convolution, we then modify the output values in which the nth element of the input is included.

To minimize number of operations in Toom-Cook algorithm we construct the polynomial $M(a) = \prod_i m_i(a)$ such that $\deg(M(a)) = \deg(s(a)) + 1$. We can further reduce the number of operations by using $M^{'}(a)$ where $\deg(M'(a)) = \deg(M(a)) - 1 = \deg(s(a))$.

Then if we apply now CRT in place of the polynomial $s(a) = w(a)x(a)$ we obtain the polynomial

$$s^{'}(a) = s(a) \mod M^{'}(a) \tag{3.7}$$

Because all m_i we use are monic (see Section 2.2.1), $M^{'}(a)$ is also monic. We have $s(a) = s^{'}(a) + R\ M^{'}(a)$, where the scalar R is the coefficient of the variable with the highest degree in $s(a) = w(a)x(a)$ i.e. $R = w_k x_n$. Finally, we have

$$s(a) = s^{'}(a) + w_k x_n M^{'}(a) \tag{3.8}$$

where $s^{'}(a)$ is a solution of the convolution with the input size equal to $n-1$. Formally, we use the notation $M(a) = M^{'}(a)(a - \infty)$ (Blahut 2010).

With this approach we need only $n - 1$ root points to construct polynomial $M^{'}(a)$ instead of n root points used to construct $M(a)$ (Section 3.1) . The comparision of number of operations needed to compute convolution with Toom-Cook and modified Toom-Cook algorithms are presented in Table 3.1 and Table 3.2.

3.2.1 Matrices construction

Let us denote matrices constructed by Toom-Cook algorithm for input n as $\mathbf{G^{(n)}}$, $\mathbf{B^{(n)T}}$, $\mathbf{A^{(n)T}}$ and for the modified Toom-Cook algorithm with input n as $\mathbf{G^{modify(n)}}$, and $\mathbf{B^{modify(n)}}$ and $\mathbf{A^{modify(n)}}$. The modified Toom-Cook algorithm for input n proceeds as follows

- Construct matrices $\mathbf{A^{(n-1)T}}$, $\mathbf{G^{(n-1)}}$ and $\mathbf{B^{(n-1)T}}$ as for Toom-Cook for the problem of size $n - 1$ with polynomial $M'(a)$.

Table 3.1: Number of multiplications for Toom-Cook and modified Toom-Cook algorithms in one-dimension for the kernel of the size 3, the output of the size m in range $(2, 8)$ and the input of the size $n = m + k - 1$.

	$\mathbf{G}$		$\mathbf{B}^T$		$\mathbf{A}^T$		All	
m	T-C	mod T-C	T-C	mod T-C	T-C	mod T-C	T-C	mod T-C
2	12	10	16	13	8	7	36	30
3	15	13	25	21	15	13	55	47
4	18	16	36	31	24	21	78	68
5	21	19	49	43	35	31	105	93
6	24	22	64	57	48	43	136	122
7	27	25	81	73	63	57	171	155
8	30	28	100	91	80	73	210	192

Table 3.2: Number of multiplications for Toom-Cook and modified Toom-Cook algorithms in two-dimensions for the kernel of the size 3, the output of the size m in range(2,8) and the input of the size $n = m + k - 1$.

	$\mathbf{G}$		$\mathbf{B}^T$		$\mathbf{A}^T$		All	
m	T-C	mod T-C	T-C	mod T-C	T-C	mod T-C	T-C	mod T-C
2	84	70	128	104	48	40	260	214
3	120	104	250	210	120	100	490	414
4	162	144	432	372	240	204	834	720
5	210	190	686	602	420	364	1316	1156
6	264	242	1024	912	672	592	1960	1746
7	324	300	1458	1314	1008	900	2790	2514
8	390	364	2000	1820	1440	1300	3830	3484

- Construct the matrix $\mathbf{G}^{\mathbf{modify(n)}}$ by adding the nth row to the matrix $\mathbf{G}^{(\mathbf{n-1})}$. This row consists of zeros and 1 exactly at the last position. Then $\mathbf{G}^{\mathbf{modify(n)}}\mathbf{w} = \mathbf{G}^{(\mathbf{n-1})}\mathbf{w} + w_k$.

- Construct the matrix $\mathbf{A^{modify(n)}}$ in the same way by adding the nth row to the matrix $\mathbf{A^{(n-1)}}$. This row includes zeros and 1 at the last position. Then $\mathbf{A^{modify(n)}x} = \mathbf{A^{(n-1)}x} + x_n$.

- Construct the matrix $\mathbf{B^{modify(n)}}$ by adding the nth row and nth column to the matrix $\mathbf{B^{(n-1)}}$. The last row includes zeros and 1 at the last position. The last column includes consecutive coefficients of the polynomial $M^{'}(a)$. Then

$$\mathbf{B^{modify(n)}}(\mathbf{G^{modify(n)}w} \odot \mathbf{A^{modify(n)}x}) = \mathbf{B^{(n-1)}}(\mathbf{G^{(n-1)}w} \odot \mathbf{A^{(n-1)}x}) + w_k x_n M^{'}(a)$$

ALGORITHM 6: Algorithm to construct transformation matrices for modified Toom-Cook algorithm.

Input: m - size of output,
k - size of kernel,
$\{p_1, \cdots, p_{n-1}\}$ - set of $n-1$ pairwise different root points.
$\mathbf{A^{(n-1)T}}$, $\mathbf{B^{(n-1)T}}$, $\mathbf{G^{(n-1)}}$ - matrices constructed by Toom-Cook algorithm for $n-1$ root points
Output: Three matrices $\mathbf{A^{modify(n)T}}$, $\mathbf{G^{modify(n)}}$ $\mathbf{B^{modify(n)T}}$ for modified Toom-Cook convolution

$n = m + k - 1$;
for $i = 1$ *to* $n-1$ **do**
  $N_i = \frac{1}{\prod_{j \neq i}(p_i - p_j)}$;
  M_i = coefficient of the polynomial $\prod_l (a - p_l)$ of ith term
end
$\mathbf{A^{modify(n)T}_{1:m,1:n-1}} = \mathbf{A^{(n-1)T}}$;
for $i = 1$ *to* $m-1$ **do**
  $\mathbf{A^{modify(n)T}_{i,n}} = 0$
end
$\mathbf{A^{modify(n)T}_{m,n}} = 1$;
$\mathbf{G^{modify(n)}_{1:k,1:n-1}} = \mathbf{G^{(n-1)}}$;
for $j = 1$ *to* $k-1$ **do**
  $\mathbf{G^{modify(n)}_{n,j}} = 0$
end
$\mathbf{G^{modify(n)}_{n,k}} = 1$;
$\mathbf{B^{modify(n)T}_{1:n-1,1:n-1}} = \mathbf{B^{(n-1)T}}$;
for $i = 1$ *to* $n-1$ **do**
  $\mathbf{B^{modify(n)T}_{i,n}} = 0$
end
for $j = 1$ *to* n **do**
  $\mathbf{B^{modify(n)T}_{n,j}} = M_j$
end

The general forms of matrices obtained by the modified Toom-Cook algorithm are presented below:

$$\mathbf{G^{modify(n)}} = \begin{bmatrix} & \mathbf{G^{(n-1)}} & \\ 0 & \cdots & 0 & 1 \end{bmatrix} \qquad \mathbf{A^{modify(n)}}^T = \begin{bmatrix} & & 0 \\ & \mathbf{A^{(n-1)T}} & \vdots \\ & & 0 \\ & & 1 \end{bmatrix}$$

$$\mathbf{B^{modify(n)}}^T = \begin{bmatrix} & \mathbf{B^{(n-1)T}} & & \mathbf{0} \\ M_1(a) & \cdots & M_n & 1 \end{bmatrix}$$

Chapter 4

Theoretical error analysis

In this chapter, we present a formal error analysis of the both Toom-Cook convolution algorithm Algorithm 5 and its modified version Algorithm 6 which, to our knowledge is the first such formulation. Our approach uses the Higham (Higham 2002) method of floating point error estimation (see Section 2.3.2) and results on the instability of Vandermonde systems by Higham (Higham 2002) and Pan (Pan 2016). Notice that we are interested in floating point error of the algorithm regardless of the error of the input data. That is why we only consider cumulation of representation errors of the algorithm parameters and errors that arise from algorithm operations.

We formulate the error bounds for Toom-Cook convolution using similar techniques to those used for another bilinear problem: the fast matrix multiplication, error estimation by Bini and Lotti (Bini and Lotti 1980), Demmel et al. (Demmel et al. 2007) and Ballard (Ballard et al. 2016). The error estimation allows us to show that the Toom-Cook convolution algorithm as well as its modified version are unstable and to identify the components of the error.

4.1 Floating point error in transformations

The core operation in the linear transformations is a matrix-vector product, which can be represented as a set of dot products $\mathbf{a}^T\mathbf{x}$. Let us take an input vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. As we are interested in error that arise from

convolution algorithm Algorithm 3 not in input data representation, we assume that $x_i \in F$, $\forall\ i = 1, 2, \ldots, n$ (Higham 2002). We also take another vector $\mathbf{a} = [a_1, a_2, \ldots, a_n]$ which is a parameter of the convolution algorithm (column or row of transformation matrices) Algorithm 3, so we include the representation error of the components of the vector $\mathbf{a}$ in our analysis. Then $fl(\mathbf{a}) = [fl(a_1), fl(a_2), \ldots, fl(a_n)]$ and $fl(a_i) = a_i(1 + \delta_i)$, where $|\delta_i| \leq \varepsilon$ $\forall\ i = 1, 2, \ldots, n$ (Higham 2002). The value ε is referred to as machine epsilon and depends on the precision of the number system (see Section 2.3). The error of the dot product computations is then equal to the absolute of the difference between ground truth and the computed solution. In theoretical analysis by ground truth we mean the exact result of computations excluding all representation and operation errors.

$$|\mathbf{a}^T\mathbf{x} - fl(fl(\mathbf{a}^T)\mathbf{x})| \leq |\mathbf{a}^T|\,|\mathbf{x}|\,\alpha^{(n)}\varepsilon + O(\varepsilon^2) \tag{4.1}$$

where $|\mathbf{x}|$ - the absolute value of the vector $\mathbf{x} = [x_1, \ldots, x_n]$ is a vector with absolute values of the entries $\mathbf{x}$, so $|\mathbf{x}| = [|x_1|, \ldots, |x_n|]$ (see Section 2.2.1). The value of $\alpha^{(n)}$ stands for the coefficent in floating point error boundary of summing up n values. If we assume the linear summation in dot product computations, $\alpha^{(n)} = n + 1$ (see (Higham 2002)). There is a wide range of summation methods that allows us to compute dot product with smaller floating point error than using linear summation. Demmel et al. (Demmel and Hida 2004) and Rump et al. (Rump, Ogita, and Oishi 2008a; Rump, Ogita, and Oishi 2008b) have analysed the error of several various summation algorithms. For generality, we do not assume any particular method of dot product evaluation. Instead, we use $\alpha^{(n)}$, which stands for the error of the dot product computations for vectors of n elements.

Also, in our analysis, the vector $\mathbf{a}$ is consider as a parameter of a convolution algorithm (Algorithm 3) not an input. The elements of the vector $\mathbf{a}$ depends on the choosen root points p_i Section 3.1.1. We write $fl(\mathbf{a})$ because the mathematically exact value of $\mathbf{a}$ may not be exactly representable in finite precision floating point number system. The error may vary for different precisions for

example for single and half floating points number systems (see Section 2.3). We want to find an estimation of algorithm error expressed with these parameters, as well as the number and the type of operations.

Note that the value of $\alpha^{(n)}$ depends on the representation error of values a_i, error from multiplication of corresponding values, the lenght of vectors (n) and the method of summation. We have three possible cases that give us different boundaries for the error of multiplication $a_i x_i$:

1. Values of a_i are not in F (the set of floating points numbers in precision under consideration). In this case we have an error from the inexact representation of a_i and from the multiplication, so $fl(a_i) = a_i(1 + \delta_i)$ where $|\delta_i| \leq \varepsilon$. Then

$$|fl(fl(a_i)x_i) - a_i x_i| \leq |a_i|\,|x_i|\,2\varepsilon + O(\varepsilon^2),\ \ \forall\, i = 1, 2, \ldots, n$$

2. Values of a_i are in F. In this case only the multiplication and summation errors remain, that is

$$|fl(fl(a_i)x_i) - a_i x_i| \leq |a_i|\,|x_i|\varepsilon + O(\varepsilon^2),\ \ \forall\, i = 1, 2, \ldots, n$$

3. Values of a_i are integer powers of 2, so we have no error either from representation or from multiplication, so

$$|fl(fl(a_i)x_i) - a_i x_i| \leq |a_i|\,|x_i|,\ \ \forall\, i = 1, 2, \ldots, n$$

If we assume linear summation in Equation (4.1) we have $\alpha^{(n)}$ for each of these three cases as follows

1. $\alpha^{(n)} = n + 1$ for any elements a_i
2. $\alpha^{(n)} = n$ for a_i exactly represented in F
3. $\alpha^{(n)} = n - 1$ if all a_i are integer powers of 2

However, $n - 1 < n < n + 1$ so using $\alpha^{(n)} = n + 1$ is a correct estimate but does not give the tightest possible bound.

4.2 Buniakowski-Schwartz inequality - observation

In this chapter we use the vector and matrix norms properties that arise from Buniakowski-Schwartz inequality Section 2.2.5 Equation (2.9). We consider the linear space $\mathbb{R}^n$ with the Euclidean inner product.

$$|\langle \mathbf{u}, \mathbf{v} \rangle| \leq \|\mathbf{u}\| \, \|\mathbf{v}\|$$

Let us notice that

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_i |u_i v_i|$$

$$|\langle -\mathbf{u}, -\mathbf{v} \rangle| = |\langle \mathbf{u}, \mathbf{v} \rangle| = |\langle -\mathbf{u}, \mathbf{v} \rangle| = |\langle \mathbf{u}, -\mathbf{v} \rangle|$$

(Inner product properties see Section 2.2.5)
So, we can replace $|\angle \mathbf{u}, \mathbf{v}\rangle|$ by $\langle |\mathbf{u}|, |\mathbf{v}| \rangle$
Finally we have

$$|\langle \mathbf{u}, \mathbf{v} \rangle| = \|\mathbf{u} \odot \mathbf{v}\|_1 \leq \|\mathbf{u}\|_2 \, \|\mathbf{v}\|_2$$

Analougously for matrices $\mathbf{U}, \mathbf{W} \in \mathbb{R}^{n \times m}$ it holds

$$\|\mathbf{U} \odot \mathbf{W}\|_1 \leq \|\mathbf{U}\|_2 \|\mathbf{W}\|_2$$

4.3 Norms equivalence

In Toom-Cook error estimation we use widely known norm property so called norms equivalence (see Definition 9 and Definition 10).

Theorem 4. *Any two norms defined on the same vector space $\mathbb{R}^n$ are equivalent.*

Proof. Based on (Rudin 1986)
Let $\|\cdot\|$ denotes a norm in $\mathbb{R}^n$. We show that it is equivalent to norm$\|\cdot\|_1$ defined in Equation (2.1).. Then transitivity of the norm equivalency gives the thesis we want.
Let us denote the canonical base in $\mathbb{R}^n$ by $\mathbf{e_1}, \ldots, \mathbf{e_n}$, that is $\mathbf{e_i} = [0, ..., 1, ...0]$ for $i = 1, ..., n$.

For any $\mathbf{x} = [x_1, ..., x_n] \in \mathbb{R}^n$ we have:

$$\|\mathbf{x}\| = \|\sum x_i \mathbf{e_i}\| \leq \sum |x_i| \|\mathbf{e_i}\| \leq \sum (|x_i| \cdot \sum \|\mathbf{e_i}\|) = \sum \|\mathbf{e_i}\| \cdot \|\mathbf{x}\|_1$$

If we put $c_1 = \sum \|\mathbf{e_i}\|$ then we obtain $\|\mathbf{x}\| \leq c_1 \|\mathbf{x}\|_1$ for any vector $\mathbf{x} \in \mathbb{R}^n$.
To obtain the second inequality let us consider a unit sphere $S = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{y}\|_1 = 1\}$. The defined set S is closed and bounded (a compact set). Define on S a function $\phi : S \longrightarrow (0, \infty)$ putting $\phi(\mathbf{y}) = \|\mathbf{y}\|$. The following inequality

$$|\phi(\mathbf{y}) - \phi(\mathbf{y_0})| = |\|\mathbf{y}\| - \|\mathbf{y_0}\|| \leq \|\mathbf{y} - \mathbf{y_0}\| \leq c_1 \|\mathbf{y} - \mathbf{y_0}\|_1$$

implies that ϕ is a continuous function. Now, from the Weierstrass theorem it follows that ϕ achieves the minimal value on S. Denote it by c_2. Hence for any $\mathbf{x} \in S$ we have $\|\mathbf{y}\| \geq c_2$. Finally for any vector $\mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x} \neq 0$ a vector $\mathbf{y} = \mathbf{x}/\|\mathbf{x}\|_1 \in S$. So $\mathbf{x} = \|\mathbf{x}\|_1 \cdot \mathbf{y}$ and $\|\mathbf{x}\| = \|\mathbf{x}\|_1 \cdot \|\mathbf{y}\| \geq c_2 \|\mathbf{x}\|_1$ what finishes the proof. □

Analogically one can prove equivalence of all matrix norms in $\mathbb{R}^{m \times n}$. However the vector spaces $\mathbb{R}^{m \times n}$ and $\text{vec}(\mathbb{R}^{m \times n}) = \{\text{vec}(W) : W \in \mathbb{R}^{m \times n}\}$ are isomorphic and so the proved above theorem could be applied to the matrix norms directly.

Based on the Theorem 4 all matrix norms introduced in Section 2.2.4 are equivalent. In particular it is true for norms $\|\cdot\|_2$ and $\|\cdot\|_F$ thus

$$\|\mathbf{W}\|_2 \leq \|\mathbf{W}\|_F \tag{4.2}$$

And for norms $\|\cdot\|_1$ and $\|\cdot\|_2$

$$\frac{1}{\sqrt{n}} \|W\|_1 \leq \|W\|_2 \leq \sqrt{n} \|W\|_1 \tag{4.3}$$

The proof of above inequalities can be found in Appendix A.

4.4 Toom-Cook error analysis

The Toom-Cook method generates algorithms for fixed-size convolutions, which are expressed as a set of three matrices, $\mathbf{G}$, $\mathbf{B}^T$ and $\mathbf{A}^T$. These matrices are computed once in advance, and can be used with many different inputs and kernels. Figure 4-1 shows the three steps of the algorithm: (a) linear transformations of the kernel $\mathbf{w}$, and the input $\mathbf{x}$; (b) element-wise multiplication between the elements of the transformed the input and the kernel (Hadamard product); and (c) the output linear transformation. All of these operations have an impact on the accuracy of the result.

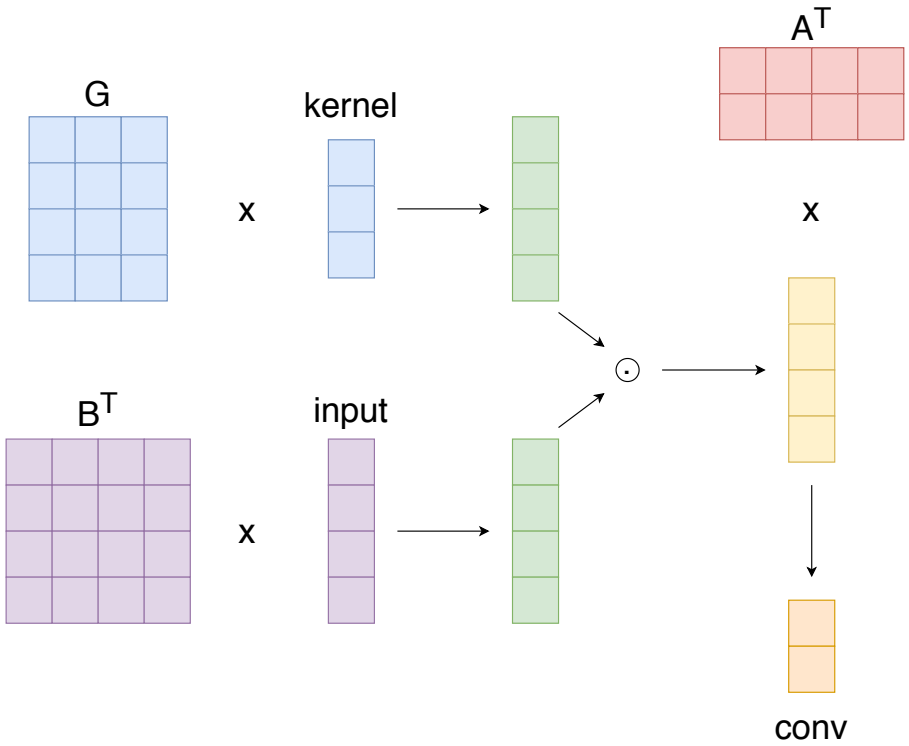


Figure 4-1: One-dimensional Toom-Cook algorithm.

In our analysis, we use properties of square Vandermonde matrices (Pan 2016) to understand the numerical properties of the Toom-Cook algorithm and conditioning of the underlying calculation. The matrices $\mathbf{G}$ and $\mathbf{A}$ as described in Chapter 3 in Algorithms 5 and 6 are rectangular. However, we can interpret these matrices as square Vandermonde matrices, which are multiplied by vectors whose last entries are equal to zero. Thus, we can analyse Toom-Cook

covolution algorithm like we used the square matrices while the implementation is actually done using rectangular matrices.

The matrix $\mathbf{G}$ shown in Figure 4-1 has only three elements in each row, rather than four, because the kernel in the example has just $k = 3$ elements. The full (square) Vandermonde matrix $\mathbf{G}$ actually has four elements per row, with the fourth element computed in the same pattern as the first three. The kernel $\mathbf{w}$ also has four elements, but the fourth is zero. Thus, the fourth element of each row of $\mathbf{G}$ is multiplied by the fourth element of $\mathbf{w}$ which is always zero. As a result, we can safely eliminate the last column of the square Vandermonde matrix $\mathbf{G}$, and crucially, all associated computation. Similarly, $\mathbf{A}^T$ in Figure 4-1 is shown with just two rows rather than four, because in this example we compute an output block of the size two (that is the number of computed convolution output values). However, we could equally show all four rows of the Vandermonde matrix $\mathbf{A}$ and discard two of the computed results.

To estimate the error bounds we will use the matrix norms $\|\cdot\|_1$, $\|\cdot\|_2$ and $\|\cdot\|_F$ (see Section 2.2.4).

We define $\alpha^{(n)}$, $\beta^{(n)}$ and $\gamma^{(k)}$ as constants in dot product floating point error bounds in Equation (4.1) for matrices $\mathbf{A}^T$, $\mathbf{B}^T$ and $\mathbf{G}$ respectively.

4.4.1 Toom-Cook error analysis in one dimension

Theorem 5. *An error for one-dimensional Toom-Cook convolution computation satisfies the normwise bound equal to*

$$\|\hat{\mathbf{s}}-\mathbf{s}\|_1 \leq \|\mathbf{A}^T\|_1 \|\mathbf{G}\|_F \|\mathbf{w}\|_2 \|\mathbf{B}^T\|_F \|\mathbf{x}\|_2 \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2) \quad (4.4)$$

An error for the qth element of one-dimensional Toom-Cook convolution computation for $q = 1, \cdots, m$ satisfies the bound equal to

$$|\hat{s}_q - s_q| \leq |\mathbf{A}^T| \left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right) \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2) \quad (4.5)$$

where values of $\alpha^{(n)}$, $\beta^{(n)}$ and $\gamma^{(k)}$ depend on the method of summation in dot product computations in matrices $\mathbf{A}^T$, $\mathbf{B}^T$ and $\mathbf{G}$ as in Equation (4.1) and ε depends on the precision of the number system used for computations.

Proof. Let $f(\mathbf{w}, \mathbf{x})$ be the bilinear function computing Toom-Cook convolution $f : \mathbb{R}^k \times \mathbb{R}^n \to \mathbb{R}^m$ such that

$$f(\mathbf{w}, \mathbf{x}) = \mathbf{A}^T(\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x})$$

The computation consists of the following

1. Kernel and input transformations:

$$f_1^w : \mathbb{R}^k \to \mathbb{R}^n \quad f_1^w(\mathbf{w}) = \mathbf{G}\mathbf{w}$$

$$f_1^x : \mathbb{R}^n \to \mathbb{R}^n \quad f_1^x(\mathbf{x}) = \mathbf{B}^T\mathbf{x}$$

2. Hadamard product:

$$f_2 : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \quad f_2(\mathbf{b}, \mathbf{c}) = \mathbf{b} \odot \mathbf{c}$$

3. Postprocessing transformation:

$$f_3 : \mathbb{R}^n \to \mathbb{R}^m \quad f_3(\mathbf{a}) = \mathbf{A}^T\mathbf{a}$$

We therefore need to find the error for the composition of these three computations, that is the error of composed function $f(\mathbf{w}, \mathbf{x})$ where

$$f(\mathbf{w}, \mathbf{x}) = f_3(f_2(f_1^w(\mathbf{w}), f_1^x(\mathbf{x})))$$

We follow Higham's method (Higham 2002) for estimating the floating point error of the result of the composed function computations. Let us denote

$$\mathbf{a_1} = [\mathbf{w}, \mathbf{x}] \qquad \text{and} \qquad \mathbf{a_{t+1}} = f_t(\mathbf{a_t})$$

that is the result of the first and $(t+1)$th stage of the algorithm.
The vector $\mathbf{a_2}$ includes preprocessing transformations of kernel and input $f_1^w(\mathbf{w}) = \mathbf{Gw}$ and $f_1^x(\mathbf{x}) = \mathbf{B}^T\mathbf{x}$.

$$\mathbf{a_2} = \begin{bmatrix} \mathbf{Gw} \\ \mathbf{B}^T\mathbf{x} \end{bmatrix}$$

The vector $\mathbf{a_3}$ is the Hadamard product of the two vectors $\mathbf{Gw}$ and $\mathbf{B}^T\mathbf{x}$ and is equal to $f_2(\mathbf{Gw}, \mathbf{B}^T\mathbf{x})$

$$\mathbf{a_3} = \mathbf{Gw} \odot \mathbf{B}^T\mathbf{x}$$

Finally, $\mathbf{a_4}$ is the post processing transformation $f_3(\mathbf{Gw} \odot \mathbf{B}^T\mathbf{x})$

$$\mathbf{a_4} = \mathbf{A}^T(\mathbf{Gw} \odot \mathbf{B}^T\mathbf{x})$$

The computed values are denoted by

$$\hat{\mathbf{a}}_{\mathbf{t+1}} = f_t(a_t) + \Delta a_{t+1}$$

where Δa_{t+1} is the floating point error of the tth stage of the algorithm which we compute using (Equation (4.1)).

Let vector $\mathbf{s} = f_3(f_2(f_1^w(\mathbf{w}), f_1^x(\mathbf{x})))$ be an exact (theoretical) result and $\hat{\mathbf{s}}$ be the computed solution and J_t is the Jacobian matrix of the function f_t. Following (Higham 2002) the computed result $\hat{s}$ is equal to

$$\hat{\mathbf{s}} = f_3(f_2(f_1^w(\mathbf{w}), f_1^x(\mathbf{x}))) + J_3J_2\Delta a_2 + J_3\Delta a_3 + \Delta a_4$$

where

$$J_3 = \mathbf{A}^T, \qquad J_2 = \big[\mathrm{Diag}(\mathbf{B}^T\mathbf{x}), \mathrm{Diag}(\mathbf{Gw})\big]$$

The floating point errors on each stage of the Winograd convolution algorithm are bounded as follows

$$
\begin{aligned}
|\Delta a_2| &\leq \begin{bmatrix} |\mathbf{G}|\,|\mathbf{w}|\,\gamma^{(k)}\,\varepsilon + O(\varepsilon^2) \\ |\mathbf{B}^T|\,|x|\,\beta^{(n)}\,\varepsilon + O(\varepsilon^2) \end{bmatrix} \\
|\Delta a_3| &\leq \left(|\mathbf{G}|\,|\mathbf{w}| \odot |\mathbf{B}^T|\,|x|\right)\,\varepsilon + O(\varepsilon^2), \\
|\Delta a_4| &\leq |\mathbf{A}^T|\left(|\mathbf{G}|\,|\mathbf{w}| \odot |\mathbf{B}^T|\,|\mathbf{x}|\right)\alpha^{(n)}\,\varepsilon + O(\varepsilon^2)
\end{aligned}
$$

The componentwise error is defined as the absolute difference between exact and computed solutions (Higham 2002; Wilkinson 1994)

$$
\begin{aligned}
&|\hat{\mathbf{s}} - \mathbf{s}| \\
&= |f_3\left(f_2\left(f_1^w\left(\mathbf{w}\right), f_1^x\left(\mathbf{x}\right)\right)\right) + J_3 J_2 \Delta a_2 + J_3 \Delta a_3 + \Delta a_4 - f_3\left(f_2\left(f_1^w\left(\mathbf{w}\right), f_1^x\left(\mathbf{x}\right)\right)\right)| \\
&= |J_3 J_2 \Delta a_2 + J_3 \Delta a_3 + \Delta a_4| \leq |J_3||J_2||\Delta a_2| + |J_3||\Delta a_3| + |\Delta a_4| \\
&\leq \left[|A^T||\,\mathrm{Diag}\left(\mathbf{B}^T\mathbf{x}\right)|\,, |\mathbf{A}^T||\,\mathrm{Diag}\left(\mathbf{G}\mathbf{w}\right)|\right]|\Delta a_2| + |\mathbf{A}^T||\Delta a_3| + |\Delta a_4| \\
&\leq \left[|\mathbf{A}^T||\,\mathrm{Diag}\left(\mathbf{B}^T\mathbf{x}\right)|\,, |\mathbf{A}^T||\,\mathrm{Diag}\left(\mathbf{G}\mathbf{w}\right)|\right] \begin{bmatrix} |\mathbf{G}||\mathbf{w}|\gamma^{(k)}\varepsilon + O\left(\varepsilon^2\right) \\ |\mathbf{B}^T||\mathbf{x}|\beta^{(n)}\varepsilon + O\left(\varepsilon^2\right) \end{bmatrix} \\
&\quad + |\mathbf{A}^T|\left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right)\varepsilon + O(\varepsilon^2) + |\mathbf{A}^T|\left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right)\alpha^{(n)}\varepsilon + O\left(\varepsilon^2\right) \\
&= |\mathbf{A}^T|\left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right)(\gamma^{(k)} + \beta^{(n)})\varepsilon + |\mathbf{A}^T|\left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right)\varepsilon \\
&\quad + |\mathbf{A}^T|\left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right)\alpha^{(n)}\varepsilon + O\left(\varepsilon^2\right) \\
&= |\mathbf{A}^T|\left(|\mathbf{G}||\mathbf{w}| \odot |\mathbf{B}^T||\mathbf{x}|\right)\left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right)\varepsilon + O\left(\varepsilon^2\right)
\end{aligned}
$$

For the normwise error estimation we use induced norm $\|\cdot\|_1$ Equation (2.4) and Frobenius norm $\|\cdot\|_F$ Equation (2.6), hence

$$
\begin{aligned}
&\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \\
&\leq \|\mathbf{A}^T\left(\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}\right)\left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right)\varepsilon + O\left(\varepsilon^2\right)\|_1 \\
&\leq \|\mathbf{A}^T\left(\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}\right)\|_1\left(\alpha^{(n)} + \beta^{(n)} + \gamma^k + 1\right)\varepsilon + O(\varepsilon^2)
\end{aligned}
$$

$$\leq \|\mathbf{A}^T\|_1 \|\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}\|_1 \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

Applying the Buniakowski-Schwartz inequality (see Section 4.2) to componentwise multiplication yields

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq \|\mathbf{A}^T\|_1 \|\mathbf{G}\mathbf{w}\|_2 \|\mathbf{B}^T\mathbf{x}\|_2 \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

As matrix norm $\|\cdot\|_2$ is induced by the vector norm $\|\cdot\|_2$ (see Definition 8) $\|\mathbf{G}\mathbf{w}\|_2 \leq \|\mathbf{G}\|_2 \|\mathbf{w}\|_2$ (see Equation (2.3)). From norms equivalence (see Equation (4.2)) $\|\mathbf{G}\|_2 \leq \|\mathbf{G}\|_F$. Finally, we have

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq \|\mathbf{A}^T\|_1 \|\mathbf{G}\|_F \|\mathbf{w}\|_2 \|\mathbf{B}^T\|_F \|\mathbf{x}\|_2 \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

□

As $\|\mathbf{A}\|_1 \leq \sqrt{n}\|\mathbf{A}\|_2 \leq n\|\mathbf{A}\|_1$ (see Definition 10 and Equation (4.3)), we have

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq \sqrt{n}\|\mathbf{A}^T\|_F \|\mathbf{G}\|_F \|\mathbf{w}\|_2 \|\mathbf{B}^T\|_F \|\mathbf{x}\|_2 \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)}\right) \varepsilon + O(\varepsilon^2)$$

Corollary 1. *If we use linear summation in the dot product computations and arbitrary values for the elements of matrices $\mathbf{A}^T$, $\mathbf{G}$ and $\mathbf{B}^T$ the componentwise bound is equal to:*

$$|\hat{\mathbf{s}} - \mathbf{s}| \leq |\mathbf{A}^T| \left(|\mathbf{G}|\,|\mathbf{w}| \odot |\mathbf{B}^T|\,|\mathbf{x}|\right) (k + 2n + 4)\varepsilon + O(\varepsilon^2)$$

and the normwise bound is equal to:

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq \sqrt{n}\|\mathbf{A}^T\|_2 \|\mathbf{G}\|_F \|\mathbf{w}\|_2 \|\mathbf{B}^T\|_F \|\mathbf{x}\|_2 (k + 2n + 4)\,\varepsilon + O(\varepsilon^2)$$

where k is the kernel size and n is the input size of the convolution

4.4.2 Toom-Cook error analysis in two dimensions

Two-dimensional convolution can be implemented by nesting one-dimensional convolutions (Blahut 2010; Lavin and Gray 2016). This nesting approach requires additional pre post-processing linear transformations. In two-dimensional convolution we use the properties of Kronecker product that means

$$\operatorname{vec}(\mathbf{M}\mathbf{X}\mathbf{M}^T) = (\mathbf{M} \otimes \mathbf{M}) \operatorname{vec}(\mathbf{X}) \tag{4.6}$$

where $\operatorname{vec}(\cdot)$ is a vector constructed from matrix by stacking its columns into a one column vector.

Notice that despite both formulas in Equation (4.6) are mathematically equivalent the result in floating point arithmetic could be different.

For two-dimensional Toom-Cook convolution algorithm we can formulate the theorem analogous to Theorem 5

Theorem 6. *An error for two-dimensional Toom-Cook convolution computation satisfies the componentwise bound equal to:*

$$|\hat{\mathbf{S}} - \mathbf{S}| \leq |\mathbf{A}^T| \left(|\mathbf{G}| \, |\mathbf{W}| \, |\mathbf{G}^T| \odot |\mathbf{B}^T| \, |\mathbf{X}| \, |\mathbf{B}| \right) |\mathbf{A}| \, R\,\varepsilon + O(\varepsilon^2) \tag{4.7}$$

An error for two-dimensional Toom-Cook convolution computation satisfies the normwise bound equal to

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq \|\mathbf{A}^T\|_1 \, \|\mathbf{G}\|_F \, \|\mathbf{W}\|_F \, \|\mathbf{G}^T\|_F \, \|\mathbf{B}^T\|_F \, \|\mathbf{X}\|_F \, \|\mathbf{B}\|_F \, \|\mathbf{A}\|_1 \, R\,\varepsilon + O(\varepsilon^2) \tag{4.8}$$

where $R = 2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1$.

We assume the identical method of summation for matrix and transpose matrix multiplication, where $\alpha^{(n)}$, $\beta^{(n)}$, $\gamma^{(k)}$ *represent errors from multiplication by matrices* $\mathbf{A}^T$, $\mathbf{B}^T$ *and* $\mathbf{G}$ *respectively.*

Proof. Let $g(\mathbf{W}, \mathbf{X})$ be a bilinear function computing Toom-Cook two dimensional convolution $g : \mathbb{R}^{k \times k} \times \mathbb{R}^{n \times n} \to \mathbb{R}^{m \times m}$ such as

$$g(\mathbf{W}, \mathbf{X}) = \mathbf{A}^T \left(\mathbf{G}\mathbf{W}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}\mathbf{B} \right) \mathbf{A}$$

The computations consist of

1. Kernel and input transformations:

$$g_1^W : \mathbb{R}^{k\times k} \to \mathbb{R}^{n\times n}, \;\; g_1^W(\mathbf{W}) = \mathbf{G}\mathbf{W}\mathbf{G}^T$$

$$g_1^X : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}, \;\; g_1^X(\mathbf{X}) = \mathbf{B}^T\mathbf{X}\mathbf{B}$$

2. Hadamard product:

$$g_2 : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}, \;\; g_2(\mathbf{M}, \mathbf{N}) = \mathbf{M} \odot \mathbf{N}$$

3. Postprocessing transformation:

$$g_3 : \mathbb{R}^{n\times n} \to \mathbb{R}^{m\times m}, \;\; g_3(\mathbf{M}) = \mathbf{A}^T\mathbf{M}\mathbf{A}$$

We put a composed function $g(\mathbf{W}, \mathbf{X})$

$$g(\mathbf{W}, \mathbf{X}) = g_3(g_2(g_1^W(\mathbf{W}), g_1^X(\mathbf{X})))$$

Denoting the exact (theoretical) result as $\mathbf{S}$ and computed result as $\hat{\mathbf{S}}$, similarly as for one-dimensional convolution we have

$$|\operatorname{vec}(\hat{\mathbf{S}}) - \operatorname{vec}(\mathbf{S})| \leq |J_3|\,|J_2|\,|\Delta b_2| + |J_2|\,|\Delta b_3| + |\Delta b_4| \tag{4.9}$$

where

$$J_3 = \mathbf{A}^T \otimes \mathbf{A}^T, \qquad J_2 = \left[\operatorname{Diag}((\mathbf{B}^T \otimes \mathbf{B}^T)\mathbf{x}), \operatorname{Diag}((\mathbf{G} \otimes \mathbf{G})\mathbf{w})\right]$$

Floating point errors for each stage of the algorithm are bounded as follows

$$|\Delta b_2| \leq \begin{bmatrix} (|\mathbf{G}| \otimes |\mathbf{G}|)\ \operatorname{vec}(|\mathbf{W}|)\, 2\gamma^{(k)}\,\varepsilon + O(\varepsilon^2) \\ (|\mathbf{B}^T| \otimes |\mathbf{B}^T|)\ \operatorname{vec}(|\mathbf{X}|)\, 2\beta^{(n)}\,\varepsilon + O(\varepsilon^2) \end{bmatrix}$$

$$|\Delta b_3| \leq \left((|\mathbf{G}| \otimes |\mathbf{G}|)\ \mathrm{vec}(|\mathbf{W}|) \odot (|\mathbf{B}^T| \otimes |\mathbf{B}^T|)\ \mathrm{vec}(|\mathbf{X}|)\right)\ \varepsilon + O(\varepsilon^2),$$

$$|\Delta b_4| \leq$$

$$(|\mathbf{A}^T| \otimes |\mathbf{A}^T|)\left((|\mathbf{G}| \otimes |\mathbf{G}|)\ \mathrm{vec}(|\mathbf{W}|) \odot (|\mathbf{B}^T| \otimes |\mathbf{B}^T|)\ \mathrm{vec}(|\mathbf{X}|)\right) 2\alpha^{(n)}\,\varepsilon + O(\varepsilon^2)$$

Putting above formulas into the Equation (4.9) we have

$$|\,\mathrm{vec}(\hat{\mathbf{S}}) - \mathrm{vec}(\mathbf{S})| \leq |J_3|\,|J_2|\,|\Delta b_2| + |J_2|\,|\Delta b_3| + |\Delta b_4|$$

$$= |\mathbf{A}^T \otimes \mathbf{A}^T|\left[\mathrm{Diag}((|\mathbf{B}^T| \otimes |\mathbf{B}^T|)\,\mathrm{vec}(|\mathbf{X}|)), \mathrm{Diag}((|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}(|\mathbf{W}|))\right]|$$

$$\left|\begin{bmatrix} (|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}(|\mathbf{W}|)2\gamma^{(k)}\varepsilon + O(\varepsilon^2) \\ \left(|\mathbf{B}^T| \otimes |\mathbf{B}^T|\right)\mathrm{vec}(|\mathbf{X}|)|2\beta^{(n)}\varepsilon + O(\varepsilon^2) \end{bmatrix}\right|$$

$$+ \left(|\mathbf{A}^T| \otimes |\mathbf{A}^T|\right)\left((|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}(|\mathbf{W}|) \odot \left(|\mathbf{B}^T| \otimes |\mathbf{B}^T|\right)\mathrm{vec}(|\mathbf{X}|)\right)\varepsilon$$

$$+ \left(|\mathbf{A}^T| \otimes |\mathbf{A}^T|\right)\left((|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}\,(|\mathbf{W}|) \odot \left(|\mathbf{B}^T| \otimes |\mathbf{B}^T|\right)\mathrm{vec}\,(|\mathbf{X}|)\right)2\alpha^{(n)}\varepsilon + O(\varepsilon^2)$$

$$= \left(|\mathbf{A}^T| \otimes |\mathbf{A}^T|\right)\left((|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}\,(|\mathbf{W}|) \odot \left(|\mathbf{B}^T| \otimes |\mathbf{B}^T|\right)\mathrm{vec}(|\mathbf{X}|)\right)\left(2\gamma^{(k)} + 2\beta^{(n)}\right)\varepsilon$$

$$+ \left(|\mathbf{A}^T| \otimes |\mathbf{A}^T|\right)\left((|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}(|\mathbf{W}|) \odot \left(|\mathbf{B}^T| \otimes |\mathbf{B}^T|\right)\mathrm{vec}\,(|\mathbf{X}|)\right)\left(2\alpha^{(n)} + 1\right)\varepsilon + O(\varepsilon^2)$$

$$= \left(|\mathbf{A}^T| \otimes |\mathbf{A}^T|\right)(|\mathbf{G}| \otimes |\mathbf{G}|)\,\mathrm{vec}\,(|\mathbf{W}|) \odot \left(|\mathbf{B}^T| \otimes |\mathbf{B}^T|\right)\mathrm{vec}\,(|\mathbf{X}|)\,|\,R\varepsilon + O(\varepsilon^2)$$

where $R = 2\gamma^{(k)} + 2\beta^{(n)} + 2\alpha^{(n)} + 1$.

Changing vectors again to matrices we obtain

$$|\hat{\mathbf{S}} - \mathbf{S}| \leq$$

$$|\mathbf{A}^T|\left(|\mathbf{G}|\,|\mathbf{W}|\,|\mathbf{G}^T| \odot |\mathbf{B}^T|\,|\mathbf{X}|\,|\mathbf{B}|\right)|\mathbf{A}|\left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right)\varepsilon + O(\varepsilon^2)$$

We do the normwise error estimation in the same way as for one-dimensional convolution, as the matrix consistency and Buniakowski-Schwartz inequality

hold both for vectors and matrices (Steele 2004).

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq \|\mathbf{A}^T \left(\mathbf{G}\mathbf{W}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}\mathbf{B}\right) \mathbf{A} \left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)\|_1$$

$$\leq \|\mathbf{A}^T \left(\mathbf{G}\mathbf{W}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}\mathbf{B}\right) \mathbf{A}\|_1 \left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

As the norm $\|\cdot\|_1$ is an induced norm we use Equation (2.3)

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq$$

$$\|\mathbf{A}^T\|_1 \| \left(\mathbf{G}\mathbf{W}\mathbf{G}^T\right) \odot \left(\mathbf{B}^T\mathbf{X}\mathbf{B}\right) \|_1 \|\mathbf{A}\|_1 \left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

Applying Buniakowski-Schwartz inequality for Hadamard product (see Section 2.2.5 and Section 4.2)

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq \|\mathbf{A}^T\|_1 \|\mathbf{G}\mathbf{W}\mathbf{G}^T\|_2 \|\mathbf{B}^T\mathbf{X}\mathbf{B})\|_2 \|\mathbf{A}\|_1 \left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

Finally, from norm equivalency (see Definition 10)

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq \|\mathbf{A}^T\|_1 \|\mathbf{G}\|_F \|\mathbf{W}\|_F \|\mathbf{G}^T\|_F \|\mathbf{B}^T\|_F \|\mathbf{X}\|_F \|\mathbf{B}\|_F \|\mathbf{A}\|_1 R\,\varepsilon + O(\varepsilon^2)$$

$$R = 2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1$$

□

From definition (see Equation (2.6)) the Frobenius norm of any matrix $\mathbf{M}$ is equal to the Frobenius norm of matrix $\mathbf{M}^T$, so we can formulate the normwise bounds

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq$$

$$\|\mathbf{A}^T\|_1 \|\mathbf{A}\|_1 \|\mathbf{G}\|^2{}_F \|\mathbf{W}\|_F \|\mathbf{B}^T\|^2{}_F \|\mathbf{X}\|_F \left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right) \varepsilon + O(\varepsilon^2)$$

Note that from norms equivalence (see Section 4.3) we can bound

$$\|\mathbf{A}^T\|_1 \|\mathbf{A}\|_1 \leq n \|\mathbf{A}^T\|_F \|\mathbf{A}\|_F = n \|\mathbf{A}^T\|^2{}_F$$

see (Higham 2002). Then we have an error bound for two-dimensional Toom-Cook algorithm equal to

$$\begin{aligned}&\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \\ &\leq n\|\mathbf{A}^T\|^2{}_F \, \|\mathbf{G}\|^2{}_F \, \|\mathbf{B}^T\|^2{}_F \, \|\mathbf{W}\|_F \, \|\mathbf{X}\|_F \left(2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1\right)\varepsilon + O(\varepsilon^2)\end{aligned}$$

Comparing it to the one-dimensional Toom-Cook convolution bound Equation (4.5), we observe that the error bound for two-dimensionsional Toom-Cook convolution computations is approximately the square of the error of the one-dimensional algorithm.

Corollary 2. *If we use a linear summation in the dot product computations and arbitrary values for the elements of matrices A^T, G and B^T, the componentwise bound for two-dimensional Toom-Cook convolution is*

$$|\hat{\mathbf{S}} - \mathbf{S}| \leq |\mathbf{A}^T| \left(|\mathbf{G}||\mathbf{W}||\mathbf{G}^T| \odot |\mathbf{B}^T||\mathbf{X}||\mathbf{B}|\right) |\mathbf{A}| \, (2k + 4n + 7)\, \varepsilon + O(\varepsilon^2)$$

and the normwise bound is equal to

$$\begin{aligned}&\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq \\ &n\|\mathbf{A}^T\|_F^2 \, \|\mathbf{G}\|_F^2 \, \|\mathbf{W}\|_F \, \|\mathbf{G}^T\|_F \, \|\mathbf{B}^T\|_F \, \|\mathbf{X}\|_F \, \|\mathbf{B}\|_F \, \|\mathbf{A}\|_1 \, (2k + 4n + 7)\, \varepsilon + O(\varepsilon^2)\end{aligned}$$

4.5 Components of the Toom-Cook error

The Toom-Cook errors in Theorem 5 and Theorem 6 states that the bound is proportional to the product of three main factors:

- The product of the norms of the three convolution matrices $\mathbf{G}$, $\mathbf{B}^T$ and $\mathbf{A}^T$.
- The product of the norms of the input $\mathbf{x}/\mathbf{X}$ and kernel $\mathbf{w}/\mathbf{W}$
- The sum of the errors from the linear transformations $\alpha^{(n)}$, $\beta^{(n)}$ and $\gamma^{(k)}$.

Interpreting the error bounds derived in Theorem 5 and Theorem 6, we see that the relative errors are controlled by norms of the Vandermonde matrices and

the summation order. Furthermore, the errors arising from the linear transformations are polynomial, as shown in Equation (4.1).

However, it should be noted that the relative condition number may depend on $\mathbf{x}$ and $\mathbf{w}$ (see Section 4.7).

As we describe in more details in Section 3.1, matrices $\mathbf{G}$ and $\mathbf{A}$ are Vandermonde matrices (theoretically square although normally presented as rectangular) and matrix $\mathbf{B}^T$ is the inverse of the square version of $\mathbf{A}^T$. The product of the norms of a square Vandermonde matrix and its inverse grows at least exponentially with their size n (Pan 2016). Thus, our bound on the error grows at least exponentially with n.

The third component of the Toom-Cook algorithm error depends on the values of $\alpha^{(n)}$, $\beta^{(n)}$, $\gamma^{(k)}$, which means that it depends on the method of evaluation of the matrix-vector multiplication.

4.6 Multiple channels

Note that in DNNs convolution is also usually computed across multiple input channels. Both their input and kernel have the same number of channels, and separate convolutions are computed for each channel. The resulting vectors or matrices (for one- and two-dimensional convolution respectively) are summed pointwise to yield a single-channel result vector or matrix (see Section 2.1.2).

Toom-Cook convolution consists of three stages: pre-processing, element-wise multiplication (Hadamard product), and post-processing. Lavin and Gray's (Lavin and Gray 2016) DNN convolution algorithm dramatically reduces the work of post-processing for multi-channel convolution. The post-processing step is a linear transformation, so the sum of the transformed Hadamard products is equal to the transformation of the sum of the Hadamard products. Thus the post-processing transformation is applied just once after summing the Hadamard products, rather than separately for each input channel before summation.

If we compute Toom-Cook convolution over C input channels we add the results of Hadamard products $\sum_{c=1}^{C}(\mathbf{G}\mathbf{w_c} \odot \mathbf{B}^T\mathbf{x_c})$ for one-dimensional convolution and $\sum_{c=1}^{C}(\mathbf{G}\mathbf{W_c}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X_c}\mathbf{B})$ for two-dimensional convolution, using the same matrices $\mathbf{G}$ and $\mathbf{B}^T$ on every channel (see Section 2.4 - Equation (2.17), Equation (2.18)). Thus we have the error

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq \|\mathbf{A}^T\|_1 \, C \, \|\mathbf{G}\|_F \max_c \|\mathbf{w_c}\|_2 \, \|\mathbf{B}^T|_F \max_c \|\mathbf{x}_c\|_2 R\varepsilon + O(\varepsilon^2) \tag{4.10}$$

where $R = \alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + 1 + \lambda^{(C)}$, $\mathbf{w_c}$ and $\mathbf{x_c}$ are the kernel and input vectors on channel c, $\alpha^{(n)}$, $\beta^{(n)}$, $\gamma^{(k)}$ represent the dot product errors and $\lambda^{(C)}$ is the error in pointwise summation.

For two dimensions we have

$$\|\hat{\mathbf{S}} - \mathbf{S}\|_1 \leq \left(\|\mathbf{A}^T\|_1 \, C \, \|\mathbf{G}\|_F \max_c \|\mathbf{w_c}\|_2 \, \|\mathbf{G}^T\|_F \, \|\mathbf{B}^T\|_F \max_c \|\mathbf{x_c}\|_2 \, \|\mathbf{B}\|_F \, \|\mathbf{A}\|_1\right) R\,\varepsilon + O(\varepsilon^2) \tag{4.11}$$

where $R = 2\alpha^{(n)} + 2\beta^{(n)} + 2\gamma^{(k)} + 1 + \lambda^{(C)}$

When summing n inputs, the worst-case error from simply accumulating to a single variable is $O(n)$. In contrast, the pairwise summation algorithm has a worst-case error of just $O(\log_2 n)$ (Knuth 1998). Given our existing error bound for Toom-Cook convolution with multiple channels, we can formulate the effect of using pairwise summation instead of a linear summation.

Corollary 3. *The error for 1D convolution based on Theorem 5 and Theorem 6, using linear summation across channels is*

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq \|\mathbf{A}^T\|_1 \, \|\mathbf{G}\|_F \, \|\mathbf{B}^T\|_F \, \|\mathbf{w}\|_F \, \|\mathbf{x}\|_F \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + C\right) \varepsilon + O(\varepsilon^2) \tag{4.12}$$

For pairwise summation across channels, the corresponding error is

$$\|\hat{\mathbf{s}} - \mathbf{s}\|_1 \leq$$
$$\|\mathbf{A}^T\|_1 \|\mathbf{G}\|_F \|\mathbf{B}^T\|_F \|\mathbf{w}\|_2 \|\mathbf{x}\|_2 \left(\alpha^{(n)} + \beta^{(n)} + \gamma^{(k)} + \lfloor \log(C) \rfloor + 2\right) \varepsilon + O(\varepsilon^2) \tag{4.13}$$

As we can observe comparing Equation (4.12) and Equation (4.13) we have smaller overall error when we use the pairwise summation over channels than for linear summation.

4.7 Estimate of norm and conditioning

This section contains the work of Professor Kirk M. Soodhalter from the School of Mathematics, Trinity College Dublin and is put into this thesis for the sake of completeness. The presented results provide some estimates of the norm and conditioning of the Toom-Cook/Winograd convolution algorithm, namely of the product $\mathbf{A}^T \left(\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}\right)$. In what follows we firstly express the Hadamard product using a special kind of matrix product called the Khatri-Rao product (see Definition 3). The result of Khatri- Rao product of two matrices is similar to Hadamard product. However, in Hadamard product we compute each element of the resulting vector/matrix as multiplication of respective elements in input matrices. In Khatri-Rao product we compute each element of the resulting vector/matrix as the Kronecker product of respective blocks of input matrices (Golub and Loan 2013). In particular we can use blocks identical to matrix rows. Then we reformulate Hadamard product in Toom-Cook/Winograd convolution algorithm formula using Khatri-Rao and Kronecker products.

Theorem 7. *Let* $\mathbf{B}$ *and* $\mathbf{G}$ *denote Vandermonde matrices, that are used in the Toom-Cook/Winograd algorithm and and* $\mathbf{x} = [x_1, \ldots, x_n]$, $\mathbf{w} = [w_1, \ldots, x_k]$ *be vectors. The Hadamard product* $\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}$ *admits the expression*

$$\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x} = \left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right) \cdot (\mathbf{x} \otimes \mathbf{w}) .$$

Proof. Let

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_{1:}^T \\ \mathbf{g}_{2:}^T \\ \vdots \\ \mathbf{g}_{n:}^T \end{bmatrix} \qquad \text{and} \qquad \mathbf{B}^T = \begin{bmatrix} \mathbf{b}_{1:}^T \\ \mathbf{b}_{2:}^T \\ \vdots \\ \mathbf{b}_{n:}^T \end{bmatrix}$$

with $\mathbf{b}_{i:}^T = [b_{i1}, b_{i2}, \ldots, b_{i,n}]$. Then the ith entry of $\mathbf{B}^T\mathbf{x}$ is $\mathbf{b}_{i:}^T\mathbf{x} = \sum_{j=1}^n b_{ij}x_j$ and the ith entry of $\mathbf{Gw} \odot \mathbf{B}^T\mathbf{x}$ can thus be written as

$$\mathbf{g}_{i:}^T\mathbf{w} \cdot \mathbf{b}_{i:}^T\mathbf{x} = \mathbf{g}_{i:}^T\mathbf{w} \cdot \sum_{j=1}^n b_{ij}x_j = \sum_{j=1}^n b_{ij}\mathbf{g}_{i:}^T \left(x_j\mathbf{w}\right),$$

which can be rewritten as the dot product

$$\mathbf{g}_{i:}^T\mathbf{w} \cdot \mathbf{b}_{i:}^T\mathbf{x} = [b_{i1}\mathbf{g_{i:}}^T, b_{i2}\mathbf{g}_{i:}^T, \ldots, b_{in}\mathbf{g}_{i:}^T] \begin{bmatrix} x_1\mathbf{w} \\ x_2\mathbf{w} \\ \vdots \\ x_n\mathbf{w} \end{bmatrix} = \left(\mathbf{b}_{i:}^T \otimes \mathbf{g}_{i:}^T\right)(\mathbf{x} \otimes \mathbf{w}).$$

This proves the result. □

We now calculate the condition number of Toom-Cook/Winograd convolution algorithm, that is

$$\kappa(\mathbf{a}) = \frac{\|\mathbf{J}(\mathbf{a})\| \cdot \|\mathbf{a}\|}{\|f(\mathbf{a})\|};$$

see Definition 11 Equation (2.8), (Trefethen and Bau 1997)

Let us define a function y of two vector variables $\mathbf{x}$ and $\mathbf{w}$ as the Kronecker product of them, that is $y(\mathbf{x}, \mathbf{w}) = \mathbf{x} \otimes \mathbf{w}$. Now we can formulate the Jacobian for the Toom-Cook/Winograd convolution transformation as the Jacobian of the composite function:

$$\mathbf{J}(\mathbf{x}, \mathbf{w}) = \mathbf{A}^T \left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right) \mathbf{J}_y(\mathbf{x}, \mathbf{w}),$$

where $\mathbf{J}_y(\mathbf{w}, \mathbf{x})$ is the Jacobian of $y(\mathbf{w}, \mathbf{x})$. Thus the condition number for Toom-Cook/Winograd convolution algorithm with the input $\mathbf{x}$ and the kernel $\mathbf{w}$ sat-

isfies

$$\kappa(\mathbf{x},\mathbf{w}) = \frac{\left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\mathbf{J}_y(\mathbf{x},\mathbf{w})\right\| \cdot \|\mathbf{x}\otimes\mathbf{w}\|}{\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\cdot(\mathbf{x}\otimes\mathbf{w})\|}. \tag{4.14}$$

Now we can estimate the upper bound for the Toom-Cook/Winograd convolution algorithm condition number.

Theorem 8. *The condition number $\kappa(\mathbf{x},\mathbf{w})$ with respect to $\|\cdot\|_1$ admits the upper bound estimate*

$$\kappa(\mathbf{x},\mathbf{w}) \leq \sqrt{mkn}\max\left\{\|\mathbf{x}\|_1, \|\mathbf{w}\|_1\right\}\kappa_2\left(\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\right).$$

Proof. To estimate the upper bound for the condition number we firstly must get appropriate lower bound estimates for the denominator of $\kappa(\mathbf{w},\mathbf{x})$. From the vector norms equivalence Equation (A.1)

$$\left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\cdot(\mathbf{x}\otimes\mathbf{w})\right\|_1 \geq \left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\cdot(\mathbf{x}\otimes\mathbf{w})\right\|_2$$

Thus, we can estimate the denominator of the condition number formula in the following way

$$\begin{aligned}\left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\cdot(\mathbf{x}\otimes\mathbf{w})\right\|_2 &= \left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\cdot\frac{\mathbf{x}\otimes\mathbf{w}}{\|\mathbf{x}\otimes\mathbf{w}\|_2}\right\|_2 \|\mathbf{x}\otimes\mathbf{w}\|_2 \\ &\geq \min_{\substack{y\in\mathbb{R}^{nk}\\ \|y\|_2=1}} \left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\cdot y\right\|_2 \|\mathbf{x}\otimes\mathbf{w}\|_2\end{aligned}$$

Applying this to Equation (4.14) we obtain

$$\kappa(\mathbf{x},\mathbf{w}) \leq \frac{\left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\mathbf{J}_y(\mathbf{x},\mathbf{w})\right\|_1 \cdot \|\mathbf{x}\otimes\mathbf{w}\|_1}{\min\limits_{\substack{y\in\mathbb{R}^{nk}\\ \|y\|_2=1}} \|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)y\|_2 \|\mathbf{x}\otimes\mathbf{w}\|_2}. \tag{4.15}$$

Now, we can formulate the bound for the numerator of the condition number formula. Using norms equivalence Equation (4.3), we can estimate

$$\|\mathbf{x}\otimes\mathbf{w}\|_1 \leq \sqrt{nk}\,\|\mathbf{x}\otimes\mathbf{w}\|_2.$$

The induced matrix norm inequalityEquation (2.3) implies

$$\left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\mathbf{J}_y(\mathbf{x},\mathbf{w})\right\|_1 \leq \left\|\mathbf{A}^T\left(\mathbf{B}^T \otimes_{KR} \mathbf{G}\right)\right\|_1 \left\|\mathbf{J}_y(\mathbf{x},\mathbf{w})\right\|_1 .$$

Taking into account 1-norm of partial derivatives with respect to the elements of vectors $\mathbf{x}$ and $\mathbf{w}$ we obtain

$$\left\|\mathbf{J}_y(\mathbf{x},\mathbf{w})\right\|_1 = \max\left\{\left\|\mathbf{x}\right\|_1, \left\|\mathbf{w}\right\|_1\right\}.$$

Furthermore, from matrix norms equivalence Equation (4.3) we have

$$\|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})\|_1 \leq \sqrt{m}\|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})\|_2$$

Substituting into Equation (4.15)

$$\kappa(\mathbf{w},\mathbf{x}) \leq \frac{\sqrt{m}\|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})\|_2 \max\{\|\mathbf{x}\|_1, \|\mathbf{w}\|_1\}\sqrt{nk}\|\mathbf{x}\otimes\mathbf{w}\|_2}{\min\limits_{\substack{y\in\mathbb{R}^{kn}\\ \|y\|_2=1}} \|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})y\|_2\|\mathbf{x}\otimes\mathbf{w}\|_2}$$

$$= \sqrt{mkn}\max\{\|\mathbf{x}\|_1, \|\mathbf{w}\|_1\}\frac{\|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})\|_2\|\mathbf{x}\otimes\mathbf{w}\|_2}{\min\limits_{\substack{y\in\mathbb{R}^{kn}\\ \|y\|_2=1}} \|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})y\|_2\|\mathbf{x}\otimes\mathbf{w}\|_2}$$

$$= \sqrt{mkn}\max\{\|\mathbf{x}\|_1, \|\mathbf{w}\|_1\}\max_{\substack{y\in\mathbb{R}^{kn}\\ y\neq 0}} \frac{\|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})\|_2\|\mathbf{y}\|_2}{\|\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G})\mathbf{y}\|_2}$$

From condition number definition with respect to 2-norm - Definition 11, Equation (2.8) we obtain

$$\leq \sqrt{mkn}\max\{\|\mathbf{x}\|_1, \|\mathbf{w}\|_1\}\kappa_2(\mathbf{A}^T(\mathbf{B}^T \otimes_{KR} \mathbf{G}))$$

that completes the proof.

□

The condition number of the Toom-Cook/Winograd convolution algorithm is bounded by the formula which is dependent on the input, the output and the kernel sizes. Consequently, the numerical error of the computations growth ex-

ponentially with respect to the size of Vandermonde matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{G}$ that are used in the Toom-Cook/Winograd algorithm. The boundary also depends on the absolute value of input and kernel vector elements. Namely, on their 1-norms $\|\mathbf{x}\|_1$ and $\|\mathbf{w}\|_1$. In the case of pathologically large elements of vectors $\mathbf{x}$ or $\mathbf{w}$ the algorithm may exhibit even worse conditioning.

4.8 Modified Toom-Cook error analysis

Our Theorem 5 and Theorem 6 about error estimation applies to both the Toom-Cook and modified Toom-Cook algorithms. However, we can distinguish error bounds for those two algorithms. In this section, we present a floating point error analysis for the modified version of Toom-Cook and show that it gives us tighter error bounds than for Toom-Cook. As before, our error analysis is novel, but we rely on prior methods and results from Higham (Higham 2002), Demmel (Demmel et al. 2007), Pan (Pan 2016) and the work of Bini and Lotti (Bini and Lotti 1980) on the error of fast matrix multiplication. The presented bounds allow us to see the exact difference in floating point error for both algorithms.

For a modified Toom-Cook algorithm, we have some zero elements in transformation matrices that are independent of the parameters (root points) we choose. The guaranteed properties of the modified Toom-Cook algorithm is that we have $k-1$ zero elements and a single 1 in the last row of $\mathbf{G}$ matrix, $n-1$ zero elements and a single 1 in the last column of $\mathbf{B}^T$ matrix and $m-1$ zero elements and 1 in the last column of $\mathbf{A}^T$ matrix. Also, we can observe that Toom-Cook matrices for input $n-1$ are submatrices of the matrices for modified Toom-Cook for input n (see Section 3.1, Section 3.2).

Let us denote the exact result vector of modified Toom-Cook algorithm for input n as $\mathbf{s}^{\mathbf{modify}(n)}$ and convolution vector computing by modified Toom-Cook algorithm for input n by $\hat{\mathbf{s}}^{\mathbf{modify}(n)}$. Similarly let us denote the exact result vector of Toom-Cook algorithm for input n by $\mathbf{s}^{(n)}$ and computed result by $\hat{\mathbf{s}}^{(n)}$. We put the vector $\mathbf{x}^{(n)}$ as a vector of the size n and $\mathbf{x}^{(n-1)}$ as the vector of size $n-1$ where $x_i^{(n-1)} = x_i^{(n)}$ for $i = 1, \ldots, n-1$.

Theorem 9. *The componentwise error of the qth element of the output of one-dimensional modified Toom-Cook for $q = 1, \ldots, m$ is bounded by*

$$|\hat{s}_q^{\mathbf{modify}(n)} - s_q^{\mathbf{modify}(n)}| \leq$$
$$|\mathbf{A}^{(n-1)T}{}_{q:}| \left(|\mathbf{G}^{(n-1)}||\mathbf{w}| \odot |\mathbf{B}^{(n-1)T}||\mathbf{x}^{(n-1)}| \right) \left(\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1 \right) \varepsilon + O(\varepsilon^2)$$
$$\text{for} \qquad q = 1, \ldots, m-1. \tag{4.16}$$

$$|\hat{s}_m^{(modify(n))} - s_m^{modify(n)}| \leq$$
$$|\mathbf{A}_{q:}{}^{(n-1)T}| \left(|\mathbf{G}^{(n-1)}||\mathbf{w}| \odot |\mathbf{B}^{(\mathbf{n-1})T}||\mathbf{x}^{(n-1)}| + |w_k||\mathbf{B}_{n:}{}^{\mathbf{modify}(n)T}||\mathbf{x}| \right)$$
$$\left(\max \left\{ \left(\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1 \right), \left(\beta^{(n)} + 1 \right) \right\} + 1 \right) \varepsilon + O(\varepsilon^2)$$
$$\text{for} \qquad q = m. \tag{4.17}$$

Proof. We denote matrices constructed for Toom-Cook algorithm of input size n as $\mathbf{G}^{(n)}$, $\mathbf{A}^{(n)T}$ and $\mathbf{B}^{(n)T}$ and for modified Toom-Cook algorithm of input size n as $\mathbf{G}^{\mathbf{modify}(n)}$, $\mathbf{A}^{\mathbf{modify}(n)T}$ and $\mathbf{B}^{\mathbf{modify}(n)T}$.

As we can see from the modified Toom-Cook algorithm definition (see Algorithm 6) we solve the problem of size n by solving problem of size $n-1$ by Toom-Cook algorithm and modify the last element of output vector. Thus the error boundary of the first $m-1$ elements of the output vector in modified Toom-Cook convolution algorithm is equal to the boundary of Toom-Cook convolution algorithm for input size $n-1$. The computation of the last output value (the last element of the output vector) include two parts: we compute the partial result using Toom-Cook convolution for input $n-1$ and add the missing value

$$s_m{}^{modify(n)} = s_m^{(n-1)} + w_k \mathbf{B}_{n:}^{\mathbf{modify}(n)} \mathbf{x}$$

Thus

$$
\begin{aligned}
&|\hat{s}_m^{\mathbf{modify}(n)} - s_m^{\mathbf{modify}(n)}| \leq \\
&|{s_m}^{(n-1)}| \left(\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1\right)\varepsilon + O(\varepsilon^2) + \\
&\qquad\qquad |w_k|\,|\mathbf{B}_{n:}^{\mathbf{modify}(n)T}|\,|\mathbf{x}|(\beta^{(n)} + 1)\varepsilon + O(\varepsilon^2) \\
&= \left(|{s_m}^{(n-1)}| + |w_k|\,|\mathbf{B}_{n:}^{\mathbf{modify}(n)T}||\mathbf{x}|\right) \\
&\qquad\qquad \max\left\{\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1, \beta^n + 1\right\}\varepsilon + O(\varepsilon^2)
\end{aligned}
$$

□

For DNNs convolution the kernel size k is almost always greater than or equal to 3 and $k \leq n$. If we compute the convolution then from dot product error analysis in Section 4.1 we know that: $\gamma^{(k)} \geq k - 1$, $\beta^{(n-1)} \geq n - 2$ and $\alpha^{(n-1)} \geq n - 2$. In this case $\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1 \geq k + 2n - 4$, while $\beta^{(n)} + 1 \leq n + 2$. When $k \geq 3$ the maximum in Section 4.8 is always equal to $\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1$. Thus, we can formulate the following corollary

Corollary 4. *If we use the kernel size $k \geq 3$ computing for one-dimensional convolution using modified Toom-Cook algorithm then the componentwise error for the last output element is bounded as follows*

$$
\begin{aligned}
&|\hat{s}_m^{(\mathbf{modify}(n))} - s_m^{\mathbf{modify}(n)}| \leq \\
&|\mathbf{A}_{q:}^{T(n-1)}| \left(|\mathbf{G}^{(n-1)}|\,|\mathbf{w}| \odot |\mathbf{B}^{(n-1)T}|\,|\mathbf{x}^{(n-1)}| + |w_k|\,|\mathbf{B}_{n:}^{\mathbf{modify}(n)T}|\,|\mathbf{x}|\right) R\,\varepsilon + O(\varepsilon^2)
\end{aligned}
\tag{4.18}
$$

where $R = \left(\gamma^{(k)} + \beta^{(n-1)} + \alpha^{(n-1)} + 1\right)$

Fixing any elements in matrices $\mathbf{G}^{\mathbf{modify}(n)}$, $\mathbf{B}^{\mathbf{modify}(n)T}$ and $\mathbf{A}^{\mathbf{modify}(n)T}$ and linear summation for computing the dot product we have the following boundaries

$$
\begin{aligned}
|\hat{s}_m^{\mathbf{modify}(n)} - s_m^{\mathbf{modify}(n)}| \leq &(|\mathbf{A}_{q:}^{T(n-1)}|(|\mathbf{G}^{(n-1)}||\mathbf{w}| \odot |\mathbf{B}^{(n-1)T}||\mathbf{x}^{(n-1)}| + \\
&w_k||\mathbf{B}_{n:}{}^{\mathbf{modify}(n)T}||\mathbf{x}|)(k + 2n + 2)\varepsilon + O(\varepsilon^2)
\end{aligned}
$$

4.9 Toom-Cook and modified Toom-Cook errors comparison

Comparing the componentwise error of Toom-Cook (Equation (4.5)) and a modified Toom-Cook algorithm (Equation (4.16) and Section 4.8) algorithms, we observe that the error of the modified Toom-Cook algorithm is smaller than the error of the Toom-Cook algorithm. We can see from the formula of the modified Toom-Cook algorithm (Equation (4.16) and Section 4.8) that, in contrast to the unmodified Toom-Cook algorithm(Equation (4.5)) the errors do not spread uniformly over all output values (output vector elements). The idea of computing one size smaller convolution and using the pseudo-point ∞ results in a different error boundary for the last output values. Thus our comparision is split into two parts: the error comparison for first $m-1$ output values and the error comparison for the last output value.

Looking to the error formulas in Equation (4.5) and Equation (4.16) for the first $m-1$ output values we observe that the submatrices used in error estimation of the modified Toom-Cook algorithm with the input of size n are the same as in the Toom-Cook algorithm with the input of the size $n-1$. Those results from the modified Toom-Cook algorithm definition in Section 3.2 – Algorithm 6. Thus we have the same error in the modified Toom-Cook algorithm with the input n as for Toom-Cook with the input $n-1$. Since the ill-conditioning of Vandermonde matrices increase exponentially with size, the error due to the conditioning of matrices in modified Toom-Cook algorithm is significantly smaller, although still exponential.

The second factor in the formulas for the first $m-1$ output values of both algorithms is the error from floating point operations. The error introduced by the dot product has tighter boundaries for modified Toom-Cook ($\gamma^{(k)}+\beta^{(n-1)}+\alpha^{(n-1)}$) than for Toom-Cook ($\gamma^{(k)}+\beta^{(n)}+\alpha^{(n)}$), if we assume the same method of summation in both algorithms. It is clear that the worst-case error of summation of $n-1$ elements is smaller than the worst-case error of summation of n elements, that is $\beta^{(n-1)}<\beta^{(n)}$ and $\alpha^{(n-1)}<\alpha^{(n)}$.

Because both components of the error in the first $m - 1$ output values are smaller in the modified Toom-Cook, we can safely conclude that the overall worse case error in these points is smaller than in the unmodified Toom-Cook algorithm.

Chapter 5

Proposed improvements and experimental results

A convolution of any output size $m > 1$ can be decomposed into the sum of smaller convolutions. For example, a convolution with $m = 8$ can be computed as eight convolutions with $m = 1$ (i.e. direct convolution), four convolutions with $m = 2$, two convolutions with $m = 4$, or one convolution with $m = 8$. With a kernel of size $k = 3$, the total number of general multiplications for each of these decompositions will be $8 \times 3 = 24$, $4 \times 4 = 16$, $2 \times 6 = 12$ or $1 \times 10 = 10$ respectively.

The larger the size of each sub-convolution, the smaller number of overlapping values occur (see Section 2.5), so fewer general multiplications are needed to compute the total output. Unfortunately, bigger output sizes lead to larger floating point error. In fact, as we show in Chapter 4, the error grows at least exponentially with $m + k - 1$.

Tables 5.1 and 5.2 summarize the number of general multiplications per output value for different output block sizes using a selection of typical kernel sizes from real-world DNNs. Clearly, we would like to benefit from the efficiency of large output block sizes. For example, a 5×5 Toom-Cook convolution with an output block size of 12×12 uses around $14\times$ fewer general multiplications per output value than direct convolution (output of the size 1×1).

Table 5.1: Number of multiplications per single output value for one and two-dimensionals direct and Toom-Cook convolutions for kernel of the size equal to 3 and 3×3, various output sizes and number of root points equal to n.

	1D		2D	
n	output size	multiplications	output size	multiplications
0	1	3	1×1	9
4	2	2	2×2	4
5	3	1.67	3×3	2.78
6	4	1.5	4×4	2.25
7	5	1.4	5×5	1.96
8	6	1.34	6×6	1.78
9	7	1.29	7×7	1.65
10	8	1.25	8×8	1.56
11	9	1.22	9×9	1.49
12	10	1.2	10×10	1.44
13	11	1.18	11×11	1.4
14	12	1.17	12×12	1.36
15	13	1.15	13×13	1.33
16	14	1.14	14×14	1.31

The formal error analysis that has appeared in Chapter 4 is a worst-case analysis. However, even if the worst-case error is potentially very large, it is important to know something about the typical error that arises in practice. Almost all formal analyses of floating point error are worst-case analyses. For example, all the analyses in Higham's standard textbook on floating point error are worst-case estimates (Higham 2002). Studies of average case probabilistic floating point error are possible in principle, but they rely on assumptions about the distribution of errors that are difficult to verify (see Section 2.3.2).

The focus of our work is on understanding and reducing the floating point error in fast DNN convolution. So rather than deal with the many pitfalls of formal average case analysis, we make empirical measurements of the floating

Table 5.2: Number of multiplications per single output value for one and two-dimensionals direct and Toom-Cook convolutions for kernel of the size equal to 5 and 5×5, various output sizes and number of root points equal to n.

	1D		2D	
n	output size	multiplications	output size	multiplications
0	1	5	1×1	25
4	-	-	-	-
5	-	-	-	-
6	2	3	2×2	9
7	3	2.33	3×3	5.44
8	4	2	4×4	4
9	5	1.8	5×5	3.24
10	6	1.67	6×6	2.78
11	7	1.57	7×7	2.47
12	8	1.5	8×8	2.25
13	9	1.44	9×9	2.09
14	10	1.4	10×10	1.96
15	11	1.36	11×11	1.86
16	12	1.33	12×12	1.78

errors. To measure the error in Toom-Cook convolution, we first need the algorithm for a specific size, which is defined by k, m and the $n = k + m - 1$ real-valued root points that are used to sample the polynomials corresponding to the input and kernel.

5.1 Methodology

We study over 40000 of root point selections and find that they have a huge impact on the floating point error (see Section 5.2). It is not surprising as the error is proportional to the norm of transformation matrices (see Chapter 4).

When generating the $\mathbf{A}^T$, $\mathbf{G}$ and $\mathbf{B}^T$ matrices (see Algorithm 5 and Algorithm 6), we represent all values symbolically rather than as floating point numbers. This allows us to generate exact values in each element of the convolution matrices. Once the elements have been generated, we convert each value to the nearest representable floating point number. Recall that $\mathbf{A}^T$, $\mathbf{G}$ and $\mathbf{B}^T$ are constant matrices, so we compute them as accurately as possible ahead of time.

Floating point numbers are constructed as a logarithmic sampling of the real number line, and the range $(-1, 1)$ is where they are the most accurate. The values of trained DNN weights are, in practice overwhelmingly concentrated in this range. Since we are interested in differentiating the inherent error in the convolution algorithms, not just in the context of specific networks, we would like to know something about the average case error independent of any specific dataset or network. For this reason, rather than model inputs and kernels with specific distributions drawn from real networks, we model them as random variables with uniform or normal distributions in the range$(-1, 1)$. The distrtibution of the sampled values is not really important, as floating point errors are not random (Kahan 1996) and distributed in nearly the same way for uniform and normal values distribution (Dahlqvist, Salvia, and Constantinides 2019) (see Section 2.3.2).

We compute the error as the entrywise norm $\|\cdot\|_{L1}$ from the difference between the result of the convolution computations, and an approximation of the numerically correct result (ground truth) . We find our approximation of the numerically correct result using direct convolution in double precision floating point ($float64$). The Toom-Cook convolution is implemented according to the formulas Equation (3.2) and Equation (3.5). Thus the error for one and two-dimensions are computed in following way:

$$\|\mathbf{A}^T(\mathbf{G}\mathbf{w} \odot \mathbf{B}^T\mathbf{x}) - \mathbf{w} * \mathbf{x}\|_{L1}$$

$$\|\mathbf{A}^T(\mathbf{G}\mathbf{W}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}\mathbf{B})\mathbf{A} - \mathbf{W} * \mathbf{X}\|_{L1}$$

where, for two vectors: $\mathbf{a} = [a_1, \cdots, a_n]$ and $\mathbf{b} = [b_1, \cdots, b_n]$ the norm $\|\cdot\|_{L1}$ is equal to the sum of absolute of a difference between correspondend elements: $\|\mathbf{a} - \mathbf{b}\|_{L1} = \sum_i |a_i - b_i|$. Thus it is equal to the vector norm $\| \cdot \|_1$. For two matrices $\mathbf{A}$ and $\mathbf{B}$ the formula is $\|A - B\|_{L1} = \sum_{i,j} |A_{ij} - B_{ij}|$.

We found that 5000 iterations of random testing were sufficient for the average error to become stable. In all experiments we use a kernel of size 3 for one-dimensional and 3×3 for two-dimensional convolution, which are the most common sizes in real DNNs.

5.1.1 Algorithm choice

We empirically compared the numerical error of convolution algorithms generated by the Toom-Cook (Algorithm 5) and modified Toom-Cook methods (Algorithm 6). The error is susceptible to the root points that are selected. However, for a given set of root points, replacing one of them with the ∞ pseudo-point almost always reduces the error. It agrees with our theoretical analysis presented in Section 4.9. For sets of root points that result in a low error, we observed that modified Toom-Cook for kernel size 3 gave a reduction in numerical error from around 2% to around 66% for one-dimensional convolution computations and up to 90% in two-dimensional convolution computations (see Table 5.3 and Table 5.4). We can notice different result for the algorithms with output of the size 9×9 which use 11 root points. The reason is in the lack of symmentry. In modified Toom-Cook algorithm we use 0, $-\infty$, 1, -1 and 7 other root points while in Toom-Cook algorithm we use 0, -1, 1 and 8 other root points that form full symmetric set - negatives and reciprocals of 2 and 4. As we show in Section 5.2.2 using such sets of root points significantly reduces floating point error.Thus in all our empirical evaluations presented in subsequent sections we used the modified version of the Toom-Cook convolution algorithm.

Table 5.3: Comparision of Toom-Cook and modified Toom-Cook algorithms error per single output value for one-dimensional computations for kernel of the size 3, various output sizes and number of root points equal to n.

		error	
n	output size	T-C	modify T-C
4	2	6.07E-08	2.43E-08
5	3	6.18E-08	5.37E-08
6	4	9.83E-08	7.21E-08
7	5	1.14E-07	1.02E-07
8	6	2.97E-07	1.27E-07
9	7	3.2E-07	2.81E-07
10	8	7.13E-07	4.03E-07
11	9	7.58E-07	7.35E-07
12	10	1.7E-06	8.98E-07
13	11	1.81E-06	1.77E-06
14	12	2.98E-06	2.31E-06
15	13	4.51E-06	4.28E-06
16	14	1.71E-05	5.71E-06
17	15	2.19E-05	1.77E-05

5.2 Selecting root points and orders of evaluation

The Toom-Cook method gives the mathematically correct result using any sufficiently large set of distinct sampling root points. However, there is a large difference between the floating point error using different sets of root points and there is no known systematic method for selecting the best root points to minimise the error.

The root points we use have an impact on the norm of matrices $\mathbf{G}$, $\mathbf{A}^T$ and $\mathbf{B}^T$ as well as for the values of $\alpha^{(n)}$, $\beta^{(n)}$ and $\gamma^{(n_h)}$ in error formula in Chapter 4 (Theorem 5, Theorem 6 and Theorem 9).

Table 5.4: Comparision of Toom-Cook and modified Toom-Cook algorithms error per single output value for two-dimensional computations for kernel of the size 3, various output sizes and number of root points equal to n.

		error	
n	output size	T-C	modify T-C
4	2×2	3.1E-07	7.78E-08
5	3×3	2.76E-07	2.51E-07
6	4×4	5.74E-07	3.74E-07
7	5×5	7.64E-07	6.64E-07
8	6×6	4.74E-06	9.75E-07
9	7×7	5.35E-06	4.44E-06
10	8×8	2.59E-05	8.67E-06
11	9×9	2.8E-05	2.9E-05
12	10×10	1.35E-04	4.08E-05
13	11×11	1.5E-04	1.47E-04
14	12×12	3.65E-04	2.56E-04
15	13×13	8.45E-04	8.0E-04
16	14×14	1.38E-02	1.35E-03
17	15×15	1.9E-02	1.45E-02

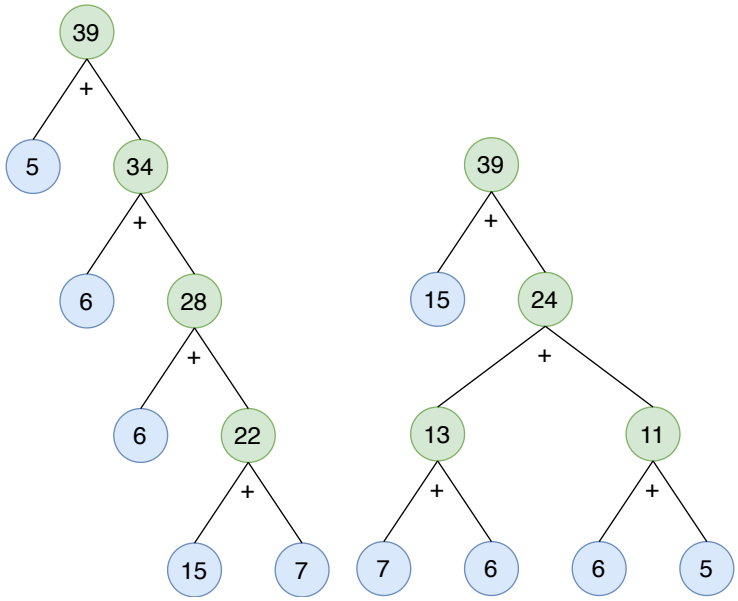
In this section, we study the problem of selecting root points experimentally. In the first stage, we simply evaluated the random sets of root points, and quickly discovered that

- Selecting an optimal set of points is NP-hard.
- Some sets of root points are much better than others.
- Not just the value of the root points, but their ordering matters. The same set of root points considered in a different order can give quite different numerical errors.

5.2.1 Canonical summation order

Different orderings of the same root points give different orderings of the values within the $\mathbf{A}^T$, $\mathbf{G}$, $\mathbf{B}^T$ matrices. The transformation steps of Toom-Cook convolution involve multiplying each of the input, kernel, and output by one of these matrices. If we change the order of entries in the matrix, then we change the order of evaluation at execution time, which causes different floating point rounding errors. Some root point orderings are better than others, but it is difficult to predict the good ones ahead of time.

Rather than searching different orderings of root points, we propose to fix the order of evaluation, so that all orderings of the same set of root points will be evaluated in the same order. The remaining problem is to pick a canonical order of evaluation that works well in practice. Each row of the $\mathbf{A}^T$, $\mathbf{G}$, $\mathbf{B}^T$ matrices is used to compute single dot product within a linear transformation, and we specify a canonical ordering for evaluating each of these dot products.



(a) Linear (b) Huffman

Figure 5-1: Linear and Huffman tree (canonical) summation methods. Our canonical ordering has two main advantages. It reduces the floating point summation error in the linear transformations by improving the order of evaluation. It also ensures that we get the same error if we evaluate the same set of root points in different orderings; the order of evaluation is determined by the Huffman tree, not by the order in which the root points are presented.

We build a Huffman (Huffman 1952) tree using the absolute values of each row, which are used to specify the order of summation. We also use simple heuristics to break ties between coefficients with the same absolute values. A basic principle of accurate floating point summation is to try to sum smaller values first, as shown in Figure 5-1. The more detailed analysis of this summation method could be found in (Higham 2002). Our Huffman tree is based purely on the values of the row elements in our transformation matrices; we build the tree at algorithm design time when the input and kernel are unknown. This makes it much easier for us to search empirically for good sets of root points because we need only to consider their value, not their ordering. Further, this method allows us to use a different order of summation for every row of matrices which is not possible to obtain by root points permutation.

A lot of different summation methods were investigated in detail by Rump et al. (Rump, Ogita, and Oishi 2008a; Rump, Ogita, and Oishi 2008b), Demmel et al. (Demmel and Hida 2004) and Castaldo et al. (Castaldo, Whaley, and Chronopoulos 2008). These methods guarantee the accurate or nearly accurate result of dot product computations. However, they require additional arithmetic operations either for a compensated summation, or to sort elements before summation, that slows down the convolution computations. These methods increased accuracy for the cost of increased computation cost, which is similar to the mixed-precision method we propose in Section 5.5.

Our canonical evaluation order is not guaranteed to sum in increasing order of absolute value because of the execution time inputs might contain big or small values. However, in practice, our canonical ordering performs much better than arbitrary orderings. We tested our approach with the setup described in Section 5.1. Across a range of convolution sizes using various root points, we found roughly a 14% improvement in accuracy for one-dimensional and 12% for two-dimensional convolutions compared with the same selection of root points in an arbitrary order. All subsequent test results presented in this chapter use our Huffman summation for the transformations.

5.2.2 Root points selection

We empirically evaluated over 40000 of random selections of values for the root points which are used to construct the $\mathbf{G}$, $\mathbf{A}^T$ and $\mathbf{B}^T$ matrices that are used to perform the linear transformations. We quickly found that it is very easy to find sets of root points that cause huge floating point errors, and more challenging to find better root points.

There is common knowledge in the literature on another approach to selecting root points to reduce the computation in the linear transformations. In general, the root points $\{0, -1, 1, \infty\}$ are useful in reducing these costs, and are used in Lavin implementation (Lavin and Gray 2016) – code. Multiplication by 1 or -1 can simply be skipped, and multiplication by zero allows both the scaling and addition to be skipped. Obviously, eliminating floating point operations also eliminates their associated error, so these root points are also suitable for reducing floating point error.

Problems start to arise when we need more than just these four basic root points. In general, selecting small, simple integers and fractions are good choices for reducing the required number of scalings and additions. We also found this type of root points to be good for reducing the floating point error. But there is no agreed-upon method in the literature for selecting between different values such as $2, -2, \frac{1}{2}, -\frac{1}{2}, \frac{3}{2}, \frac{2}{3}$, etc.

To help us find good sets of these simple values to reduce the floating point error, we developed the following rules which act as a heuristic to guide our search. The size of the kernel and output block determine the number of root points needed. We start with the basic root points $\{0, -1, 1, \infty\}$, which work well when four root points are needed. We perform our search for sets of good root points for output n based on the good sets of root points for output $n - 1$. We identify a set of potentially interesting rational root points with numerator in $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ and denominator in $\{1, 2, 3, 4\}$. This approach gives us a set P of 23 possible root points.

Root points selection algorithm

- We start with a set P_{n-1} of $n-1$ good root points $p_1, \ldots, p_{n-1}$.

- We construct sets of n root points by adding p_n ($p_n \in P$) to P_{n-1} ($P_n = P_{n-1} \cup \{p_n\}$). As we want matrices G, B^T and A^T be nonsingular therefore root point p_n must be different than all $p_i \in P_{n-1}$ ($P_{n-1} \cap \{p_n\} = \emptyset$).

- In addition:
  If n is even, then we have at least one root point $p_j \in P_{n-1}$ without symmetry, that is, $\exists p_j \in P_{n-1}$ and $-\frac{1}{p_j} \notin P_{n-1}$. We construct sets of n root points by removing the root point p_j and add instead all symmetric pairs of root points p_k and $-\frac{1}{p_k}$ that are in a set of potientially interesting root points P and are not in P_{n-1} ($P_n = P_{n-1} \backslash \{p_j\} \cup \{p_k, -\frac{1}{p_k}\}$ ($P_{n-1} \cap \{-\frac{1}{p_j}\} = \emptyset$, $P_{n-1} \cap \{p_k, -\frac{1}{p_k}\} = \emptyset$)

- If there are different sets of root points for $1D$ and $2D$ we check both of them.

- We check empirically which of the sets of root points constructed with above rules has the smallest floating point error and assign it to be P_n.

The resulting sets of "good" root points are presented in Table 5.5. The basic four root points are always $\{0, -1, 1, \infty\}$. Table 5.5 shows that when we add a fifth root point, we found empirically that $\frac{1}{2}$ is the best root point to add for both 1D and 2D convolution. Occasionally, when moving to the next larger number of root points, we remove an existing root point and add two new ones, such as when we add the eighth root point for 1 dimensional convolution. Note that the floating point error versus direct convolution grows rapidly with the number of root points. The growth in error appears to be roughly exponential in the number of root points in practice. However, the growth in error is not smooth (see Figure 5-3).

Table 5.5: Example root points for Toom-Cook in $float32$ with kernels of size 3 (for 1D) or 3×3 (for 2D). We start with set of 4 root points $P_4 = \{0, -1, 1, \infty\}$. We present the sets of root points for different cardinality as an union and/or subtraction of sets of root points.

	1D		2D	
n	Root points	output size	Root points	output size
0	Direct convolution	1	Direct convolution	1×1
4	$P_4 = \{0, -1, 1, \infty\}$	2	P_4	2×2
5	$P_4 \cup \{\frac{1}{2}\}$	3	$P_4 \cup \{\frac{1}{2}\}$	3×3
6	$P_4 \cup \{\frac{1}{2}, -3\}$	4	$P_4 \cup \{\frac{1}{2}, -2\}$	4×4
7	$P_4 \cup \{\frac{1}{2}, -\frac{1}{2}, -3\}$	5	$P_4 \cup \{\frac{1}{2}, -2, -\frac{1}{2}\}$	5×5
8	$P_8 = P_4 \cup \{\frac{1}{2}, -\frac{1}{2}, 2, -2\}$	6	P_8	6×6
9	$P_8 \cup \{-\frac{1}{4}\}$	7	$P_8 \cup \{-\frac{1}{4}\}$	7×7
10	$P_{10} = P_8 \cup \{-\frac{1}{4}, 4\}$	8	P_{10}	8×8
11	$P_{10} \cup \{\frac{1}{4}\}$	9	$P_{10} \setminus \{0\} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	9×9
12	$P_{10} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	10	$P_{10} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	10×10
13	$P_{10} \cup \{\frac{3}{4}, -\frac{4}{3}, \frac{1}{4}\}$	11	$P_{10} \cup \{\frac{3}{4}, -\frac{4}{3}, \frac{1}{4}\}$	11×11
14	$P_{14} = P_{10} \cup \{\frac{1}{4}, -\frac{3}{4}, \frac{4}{3}, -4\}$	12	P_{14}	12×12
15	$(P_{14} \setminus \{0\}) \cup \{\frac{2}{3}, -\frac{3}{2}\}$	13	$P_{14} \setminus \{0\} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	13×13
16	$P_{14} \cup \{\frac{2}{3}, -\frac{3}{2}\}$	14	$P_{14} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	14×14
17	$P_{14} \cup \{\frac{2}{3}, -\frac{3}{2}, -\frac{2}{3}\}$	15	$P_{14} \cup \{\frac{2}{3}, -\frac{3}{2}, \frac{3}{2}\}$	15×15
18	$P_{14} \cup \{\frac{2}{3}, -\frac{3}{2}, -\frac{2}{3}, \frac{3}{2}\}$	16	$P_{14} \cup \{\frac{2}{3}, -\frac{3}{2}, -\frac{2}{3}, \frac{3}{2}\}$	16×16

5.2.3 Chebyshev nodes

There is no single recognised method for selecting root points that minimize the floating point error. However, the Chebyshev nodes are known to improve the conditioning of polynomial interpolation (Higham 2002), which is an essential step of Toom-Cook convolution. Results for the floating point error of using the Chebyshev nodes can be found in Table 5.7. In general, the Chebyshev nodes are orders of magnitude better than typical random root point selections, but suboptimal for small convolution sizes.

Table 5.6: Floating point error for Toom-Cook convolution with kernel of the size 3 (1D) and 3×3 (2D) for different output size using sets of root points as presented in table Table 5.5.

	1D		2D	
n	output size	error	output size	error
0	1	1.75E-08	1×1	4.63E-08
4	2	2.45E-08	2×2	7.65E-08
5	3	5.19E-08	3×3	2.35E-07
6	4	6.92E-08	4×4	3.29E-07
7	5	9.35E-08	5×5	6.81E-07
8	6	1.15E-07	6×6	8.79E-07
9	7	2.34E-07	7×7	3.71E-06
10	8	3.46E-07	8×8	7.35E-06
11	9	5.91E-07	9×9	2.2E-05
12	10	7.51E-07	10×10	3.22E-05
13	11	1.32E-06	11×11	1.09E-04
14	12	1.84E-06	12×12	1.99E-04
15	13	3.42E-06	13×13	5.54E-04
16	14	4.26E-06	14×14	8.8E-04
17	15	1.35E-05	15×15	1.07E-02
18	16	2.24E-05	16×16	1.93E-02

5.3 Error growth

Let us consider one-dimensional convolution and the growth in error when increase from seven to eight root points. The error grows from 9.35×10^{-8} to 1.15×10^{-7}, which is a factor of around $1.12\times$. In contrast the growth in error from eight to nine root points is 1.15×10^{-7} to 2.34×10^{-7}, which is a factor of $2.03\times$. This is not a coincidence. The empirically good solution that we found when seven root points are used for one-dimensional convolution is $\{0, -\frac{1}{2}, \frac{1}{2}, -1, 1, -3, \infty\}$. In contrast when eight root points are needed, the

Table 5.7: Error for one- and two-dimensional convolution for Chebyshev points and root points we found. Column "Ratio" present how many times the error for Chebyshev points is bigger then for root points we found.

	1D			2D		
n	Our points	Chebyshev	**Ratio**	Our points	Chebyshev	**Ratio**
4	2.49E-08	3.51E-08	**1.41**	7.65E-08	1.01E-07	**1.32**
5	5.21E-08	5.535E-08	**1.06**	2.46E-07	2.12E-07	**8.62**
6	7.32E-08	1.04E-07	**1.42**	3.49E-07	5.65E-07	**1.62**
7	1.00E-07	1.68E-07	**1.68**	7.11E-07	1.70E-06	**2.39**
8	1.23E-07	3.78E-07	**3.07**	9.21E-07	6.50E-06	**7.06**
9	2.59E-07	6.76E-07	**2.61**	4.07E-06	2.59E-05	**6.36**
10	3.87E-07	1.48E-06	**3.82**	8.23E-06	1.09E-04	**1.32E+01**
11	6.62E-07	3.18E-06	**4.80**	2.53E-05	4.84E-04	**1.91E+01**
12	8.42E-07	7.46E-06	**8.86**	3.67E-05	2.18E-03	**5.94E+01**
13	1.56E-06	1.53E-05	**9.81**	1.24E-04	1.00E-02	**8.06E+01**
14	2.10E-06	3.21E-05	**1.53E+01**	2.32E-04	4.72E-02	**2.03E+02**
15	3.91E-06	7.12E-05	**1.82E+01**	6.35E-04	2.31E-01	**3.64E+02**
16	5.06E-06	1.56E-04	**3.08E+01**	1.04E-03	1.10	**1.06E+03**
17	1.68E-05	3.53E-04	**2.10E+01**	1.31E-02	5.43	**4.15E+02**
18	2.36E-05	8.03E-04	**3.40E+01**	0.24E-02	26.91	**1.12E+04**

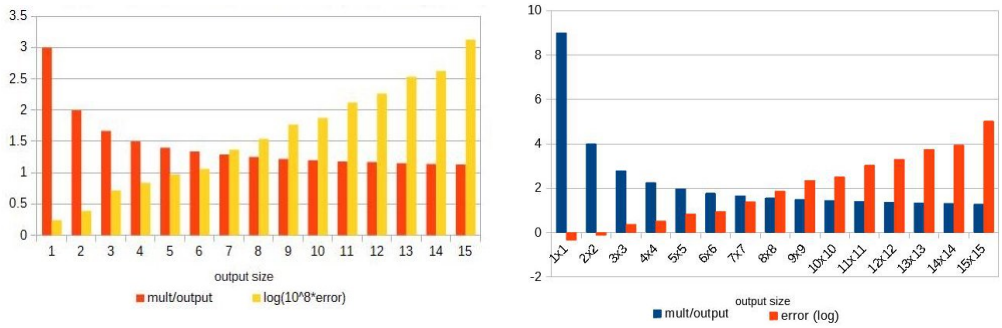
good solution is $\{0, -\frac{1}{2}, \frac{1}{2}, -1, 1, -2, 2, \infty\}$. Among the eight root points there is a symmetry between the four values $\{-\frac{1}{2}, \frac{1}{2}, -2, 2\}$, which are negations and reciprocals of one another. As we discuss in Section 5.4, these symmetries reduce a floating point error.

In contrast, where seven root points are needed, the root points $\{-\frac{1}{2}, \frac{1}{2}, -3\}$ do not cancel in the same way, and so the error for seven root points is larger than a smooth growth in error with root points would suggest. Note that the appearance of the root point -3 as the sixth selected root point for one-dimensional convolution was a great surprise to us. However, -3 has just two significant binary digits, so there is no representation error of -3 in floating point, and multiplication by -3 causes a very small error. Further, when com-

puting differences between pairs of root points for the B^T matrix, $-3-(-1)=-2$ and $-3-1=-4$, the absolute value of both of them are powers of two.

For two-dimensional convolution, small differences mean that -3 is slightly worse than $\frac{1}{2}, 2$ or -2 and is not selected.

Given these trends, it is reasonable to ask where are the good trade-offs between computation and error growth. The number of general multiplications is simply the number of root points in the convolution algorithm. Using good root point selections has no additional cost over bad ones, but dramatically reduces the error. Similarly, our Huffman summation reduces the error at no additional computation cost. There is no single best trade-off because it depends on the required accuracy. However, using our methods, it may be possible to increase the output block size by perhaps 1-3 units, without loosing accuracy.



(a) 1D (b) 2D

Figure 5-2: Number of multiplication and error for single output value for different input block sizes in one- and two-dimensional Toom-Cook convolution

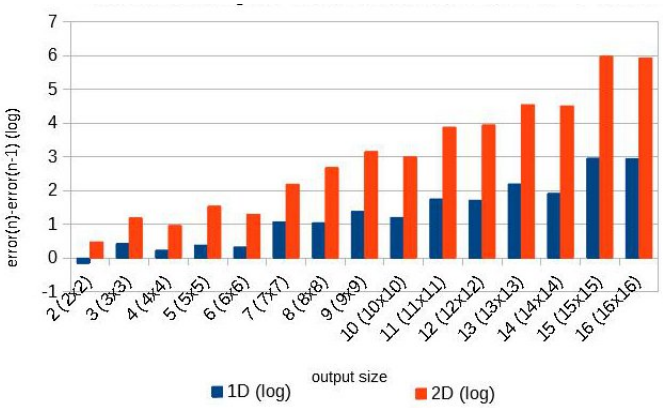


Figure 5-3: Increasing error in one- and two-dimensional Toom-Cook convolution. The vertical axis show the difference of logarithms from error of convolution computed with n and $n-1$ root points, to show error growth.

When we examine the measured errors in Figure 5-3 we see that even using good root point selections the measured error increases roughly exponentially with the size of the convolution. The average measured error grows at a rate that is compatible with the worst-case bound proven in Chapter 4.

Our analysis in Chapter 4 also suggests that the floating point error for two-dimensional convolution grows quadratically more quickly than the error for one-dimensional convolution. This is borne out in Figure 5-3, where the two-dimensional convolution error as a function of the one-dimensional convolution error is approximately $f(x) = \frac{x^2}{2.97}$.

As a further check on the consistency of our measurements, we also implemented a running error analysis for one-dimensional Toom-Cook convolution. Running error analysis (Higham 2002, p. 65) is an empirical method that computes a partial bound based on actual values alongside the executing algorithm. In our experiments we found that the running error closely matched the exponential rate of growth of the average error, with the running error $4.63\times$ to $7.51\times$ times the average error.

5.4 Discussion of root point selection

The root point selection affects both components of forward error: conditioning of transformation matrices and floating point error. The goal of our tests is to find a good balance between them. Based on the our theoretical analysis in Chapter 4, literature ((Higham 2002; Gautschi 1974; Gautschi 1990)) and our empirical experiments, it is possible to explain why some root points are better than others.

One common way to mitigate ill-conditioning of Vandermonde matrices is to use Chebyshev nodes. We tried this approach (see Section 5.2.3), but found that it did not perform well.

The size of Vandermonde matrices used in DNN convolution is relatively small. The error generated by ill-conditioning grows exponentially. But for the small convolutions in DNNs, the error from ill-conditioning is not large

enough as to outweigh the error from floating point operation and representation. Chebyshev nodes are mostly irrational, so they can not be represented exactly as floating point values. This representation error propagates throughout the algorithm.

In addition, we are interested in the accuracy of discrete convolution. There is a single correct answer (with known degree) to the interpolation in Toom-Cook convolution. If we were computing with infinite precision, we could compute the result polynomial precisely. This is somewhat different from another common use of interpolation, which is to estimate a polynomial where the degree is unknown. The advantages of Chebyshev interpolation points to mitigate Runge's phenomenon does not help in Toom-Cook convolution. We can not ignore conditioning entirely. Nevertheless, our goal is to find a set of root points which will minimize both factors: problem conditioning and floating point error.

The set of four basic root points $\{0, -1, 1, \infty\}$ are almost always the right choice. In particular, 0 and ∞ result in guaranteed zeros in all three matrices A^T, G and B^T which cause no floating point error. We note that some clear rules for root point selection emerge from our empirical study.

Firstly, we should use pairs of root points that differ in sign (positive/negative), and pairs of reciprocal root points — see Table 5.5. If we use root point p then using $-p$, $\frac{1}{p}$ and $-\frac{1}{p}$ allow us to get better accuracy than introducing another root point. Positive/negative pairs of root points generate lower elements in matrix B^T (see Algorithm 5). Examining the formula for constructing the elements of matrix B^T, we note that multiplication $(a-p)(a+p) = -p^2 + 0a + a^2$ results in a zero coefficient in the second term. Reciprocal root points introduce the coefficient of the first term equal to 1 in multiplication $(a-p)(a-\frac{1}{p}) = 1 - \frac{p^2+1}{p}a + a^2$ that does not introduce any additional scaling error. The opposite points are also known to be good for conditioning of Vandermonde matrices (Gautschi 1974; Gautschi 1990).

Secondly, the floating point error boundary depends directly on the values used in operations (see Equation (4.1)). That means that we should look for the

small values close to one to reduce the error. Putting this together with the previous observation, we can say that choosing rational root points minimizing numerator and denominator is a good strategy. This approach to root points selection also has a positive impact on the floating point error. Assuming that each of kernel and input elements have a similar distribution, then using scaling factors with a similar order of magnitude, makes it more likely to avoid cancellation error while computing dot product. An interesting point is that for bigger sets, this is not always leading rule. We found that $\frac{1}{4}$ together with $(0, -1, 1, -1/2, 1/2, -2, 2)$ works better than $\frac{1}{3}$ for 9 root points (see Tables 5.5 and 5.8). We explain this phenomenon later in this chapter.

Thirdly, the representation error of matrices elements have a big impact on the accuracy of the result. The representation error propagates through all floating point operations and therefore can grow significantly. This is why the exactly represented root points work well (see Tables 5.5 and 5.8).

Finally, as we described in Section 4.1, the error from multiplication while computing the dot product also affects accuracy. The elements equal to the power of 2 do not introduce any error from scaling and therefore keep the floating point error small. This explains why root point $\frac{1}{4}$ is better than $\frac{1}{3}$. The value $\frac{1}{4}$ in contrast to $\frac{1}{3}$ is exactly represented and does not introduce any error from multiplication.

Thus there is no simple algorithm to choose a set of good root points. Our theoretical analysis allows us to identify all components of the error and dramatically narrow the search space. With that knowledge, it is possible to check the narrowed search space empirically, to find which of the sets of root points work the best in practice.

5.5 Mixed-precision pre/post-processing

In CNNs we typically apply the same kernel to a set of many different inputs. Similarly, we compute convolution with different kernels for the same input data (see Section 2.1.2). Thus, the pre/post processing of each input, kernel and

output are done just once, whereas the transformed data is used many times. One way to improve accuracy is to use a mixed-precision algorithm, where the pre/post processing is done in higher precision, while the inner loops that perform the pairwise multiplication (Hadamard product) are computed in standard precision. This approach lowers the value of machine epsilon ε for the linear transformations in the error formula in Theorem 5 and Theorem 6. This idea may be use to run networks on embedded devices.

Table 5.8 shows the root point selection and measured errors for a mixed-precision Toom-Cook that performs the pre-processing in $float64$ and all other processing in $float32$. We found that the mixed precision algorithm reduced the error in both $1D$ and $2D$ by up to around 40% (see column "Ratio" in Table 5.8). The result is that for the same level of error, the mixed-precision algorithm can often allow an output size that is one unit larger. We observe that in most cases the same sets of root points worked best for convolution computed in $float32$ and mixed precision. Where there are differences, in most cases, this is the result of a slight difference in the order in which root points are selected when the number of root points is odd. In the mixed-precision version, rounding errors during the pre/post processing steps become a little less important because intermediate values are represented in $float64$.

5.6 Multiple channels

The proposed techniques up to this point of this chapter have been for simple one- or two-dimensional convolution with size 3 or 3×3. However, an important feature of convolution in deep neural networks is multiple channels Figure 2-4. Convolution inputs and kernels in many of the best known DNNs, such as GoogLeNet (Szegedy et al. 2015) or ResNet (He et al. 2016) typically have something between 3 and 1024 channels. However, the number of channels is a parameter selected by the designer of the neural network, and there is no upper limit on the number of channels used.

Table 5.8: Example root points for Toom-Cook transformations in $float64$ and Hadamard product in $float32$ with kernels of size 3 (for 1D) or 3×3 (for 2D). We start with set of 4 root points $P_4 = \{0, -1, 1, \infty\}$. We present the sets for different cardinallity as an union and/or substraction of sets of root points.

	1D		2D	
n	Root points	output size	Root points	output size
0	Direct convolution	1	Direct convolution	1×1
4	$P_4 = \{0, -1, 1, \infty\}$	2	P_4	2×2
5	$P_4 \cup \{3\}$	3	$P_4 \cup \{3\}$	3×3
6	$P_4 \cup \{3, -\frac{1}{2}\}$	4	$P_4 \cup \{3, -\frac{1}{2}\}$	4×4
7	$P_4 \cup \{3, -\frac{1}{2}, \frac{1}{2}\}$	5	$P_4 \cup \{3, -\frac{1}{2}, \frac{1}{2}\}$	5×5
8	$P_8 = P_4 \cup \{-\frac{1}{2}, \frac{1}{2}, -2, 2\}$	6	P_8	6×6
9	$P_8 \cup \{-\frac{1}{4}\}$	7	$P_8 \cup \{4\}$	7×7
10	$P_{10} = P_8 \cup \{-\frac{1}{4}, 4\}$	8	P_{10}	8×8
11	$P_{10} \cup \{\frac{1}{4}\}$	9	$P_{10} \setminus \{0\} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	9×9
12	$P_{12} = P_{10} \cup \{\frac{3}{4}, -\frac{4}{3}\}$	10	P_{12}	10×10
13	$P_{12} \cup \{\frac{1}{4}\}$	11	$P_{12} \cup \{-4\}$	11×11
14	$P_{14} = P_{12} \cup \{\frac{1}{4}, -4\}$	12	P_{14}	12×12
15	$P_{12} \setminus \{0\} \cup \{\frac{2}{3}, -\frac{3}{2}\}$	13	$P_{12} \setminus \{0\} \cup \{-\frac{3}{4}, \frac{4}{3}\}$	13×13
16	$P_{16} = P_{14} \cup \{\frac{2}{3}, -\frac{3}{2}\}$	14	$P'_{16} = P_{14} \cup \{-\frac{3}{4}, \frac{4}{3}\}$	14×14
17	$P_{16} \cup \{-\frac{2}{3}\}$	15	$P'_{16} \cup \{\frac{3}{2}\}$	15×15
18	$P_{18} = P_{16} \cup \{-\frac{2}{3}, \frac{3}{2}\}$	16	P_{18}	16×16

When performing convolution, a separate convolution is performed at each channel, and then the results of each separate convolution are summed with the corresponding values in the other channels (see Section 2.1.2). The obvious way to implement summation across channels is to perform the complete convolution separately on each channel and sum the results. However, this would require that the linear post-processing transformation is applied to each channel, which is a relatively expensive operation. Lavin and Gray (Lavin and Gray 2016) observed that, post-processing and summation commute because they are linear transformations on orthogonal subspaces. So, the items could be summed before the matrix multiplications so that the post-processing step

Table 5.9: The column Ratio present the ratio between the error of mixed-precision convolution and the error with all computations in $float32$.

	1D			2D		
n	output size	error	**Ratio**	output size	error	**Ratio**
0	1	1.75E-08	**1**	1×1	4.63E-08	**1**
4	2	1.87E-08	**0.76**	2×2	5.27E-08	**0.69**
5	3	3.66E-08	**0.71**	3×3	1.62E-07	**0.65**
6	4	4.41E-08	**0.64**	4×4	2.14E-07	**0.69**
7	5	6.09E-08	**0.65**	5×5	3.69E-07	**0.54**
8	6	6.97E-08	**0.61**	6×6	5.18E-07	**0.59**
9	7	1.55E-07	**0.66**	7×7	2.42E-06	**0.65**
10	8	2.09E-07	**0.6**	8×8	4.41E-06	**0.6**
11	9	3.64E-07	**0.62**	9×9	1.27E-05	**0.58**
12	10	4.50E-07	**0.6**	10×10	1.89E-05	**0.59**
13	11	8.25E-07	**0.63**	11×11	6.38E-05	**0.59**
14	12	1.11E-06	**0.6**	12×12	1.14E-04	**0.57**
15	13	2.17E-06	**0.63**	13×13	3.08E-04	**0.56**
16	14	2.78E-06	**0.65**	14×14	4.95E-04	**0.56**
17	15	8.43E-06	**0.62**	15×15	5.93E-03	**0.55**
18	16	1.39E-05	**0.62**	16×16	1.04E-02	**0.54**

need to be applied only to the sum. Therefore, in order to perform convolution over multiple channels, we have to sum up the results of pairwise multiplication (Hadamard product) before we apply the transposition represented by $\mathbf{A}^T$ matrix. The convolution is computed according to the following formulas:

$$\mathbf{A}^T(\sum_c(\mathbf{G}\mathbf{w_c} \odot \mathbf{B}^T\mathbf{x_c})) \tag{5.1}$$

$$\mathbf{A}^T(\sum_c(\mathbf{G}\mathbf{W_c}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X_c}\mathbf{B}))\mathbf{A} \tag{5.2}$$

where $c = 1, \ldots, C$ and C is a number of input channels.

As a result, the floating point error from DNN convolution is not just the error of the one- or two-dimensional convolution, but also the error from summing across channels. The impact of summation over channels in error formulas (Equation (4.10)) and (Equation (4.11)) is represented by λ. Second, there are well-known techniques for reducing the error from summation. If we reduce the error of summation, this may offset some part of the loss of accuracy arising from Toom-Cook convolution.

In Section 5.5, we proposed a mixed precision algorithm that does pre/post-processing in higher precision. However, this is not a suitable approach to increase the accuracy of the summation across channels. Summation across channels is in the inner-most loop of DNN convolution, so we cannot afford to double its cost. We instead propose the well-known pairwise summation algorithm (Knuth 1998) for summing across channels.

Tables B.1, B.4, B.7 and B.10 present measured errors for Toom-Cook convolution with just a single channel, with 32 channels, and with 64 channels.

We see that the error per output value for 64 input channels is much larger than the error for just a single input channel, but not 64 times larger. The reason is that when these values are summed, some of the errors cancel one another.

Our results presented in Tables B.2, B.3, B.5 and B.6 show that pairwise summation can reduce the total floating point error by around 20%–40%.

Similar tests presented in Tables B.8, B.9, B.11 and B.12 show that when pairwise summation across channels is used with mixed-precision transformations, the improvement compared to mixed-precision transformations alone is $25\% - 45\%$. Using both proposed methods: mixed precision and pairwise summation gives us an improvement in accuracy of around 50% in both one- and two-dimensional computations (see Table B.13).

5.7 Conclusions

In this chapter we present the methodology and empirical results of our investigations on floating point error of one- and two-dimensional Toom-Cook

convolution algorithm. We observe that using modified version of Toom-Cook convolution algorithm reduces the error by 20% to over 70%, with no additional computation cost.

We notice that the order of root point selection impacts the accuracy of the algorithm. We propose a canonical evaluation order based on Huffman trees. This fixes the order of evaluation, and empirically reduces the error by a further $12\% - 14\%$, in addition to the improvement from using the modified algorithm, again with no additional computation cost.

We identify four key criteria for good root point selections: (1) few significant mantissa bits, (2) positive/negative root point symmetry, (3) reciprocal root point symmetry, and (4) subtractions of root points leading to few significant mantissa bits. For important convolution sizes for DNNs, our empirically selected root points yield much better accuracy than the Chebyshev nodes. The Chebyshev nodes fail to meet our criteria (1), (3) and (4) which makes them relatively poor choices for the small convolutions found in DNNs.

We also propose a mixed precision approach where the transformations are computed in double precision, while the remaining inner loops are computed in $float32$. We find that performing pre/post-processing transformations in $float64$ decreases error typically by around one third in our experiments. We also empirically investigate summation across channels using pairwise summation and find that this reduces the error by around 20% to 50%. Unlike the other methods we investigated, mixed precision transformations and pairwise summation impose an additional computation cost.

Using our root point selections and techniques for improving floating points accuracy, we can reduce the error between $2\times$ and orders of magnitude, when compared with the Chebyshev nodes. Each approximately $2\times$ reduction in the error allows the output block size dimension to be increased by around one.

Chapter 6

Winograd Convolution Algorithm with Super-linear Polynomials

The family of Winograd's algorithms can be used to generate a wide variety of convolution algorithms with different trade-offs. They require a set of polynomials whose roots are used as parameters to generate transformation matrices based on Chinese Remainder Theorem (Theorem 2). These polynomials can be linear (degree d equal to 1) as described in previous Chapter 3 or super-linear (degree d greater than 1).

If only linear polynomials are used (Toom-Cook version) their roots and all transformation matrices elements belong to the field of real numbers. In this case convolution algorithms are guaranteed to need only the (theoretically specified) minimum number of general multiplication operations. The selected tile size is critical to the performance of Winograd convolution. A larger tile size increases the number of element-wise multiplication operations needed for that tile, but on the other hand they compute more results per tile. Taking into account the overlapping that is an extra operations needed at the boundary of each tile (see Section 2.1.2), larger tiles reduce the number of general multiplication operations per computed output value (see Chapter 5). However, the floating point error also grows exponentially with the tile size (Chapter 4 and Chapter 5). This is the reason why existing implementations of Winograd for DNNs typically use a small tile size.

Using higher order polynomials increases the required number of general multiplications per single output value. Thus, higher order polynomials offer a very similar trade-off as using Toom-Cook with smaller tile sizes. This chapter addresses the question: Is there a benefit in using the Winograd method with super-linear polynomials for DNNs, as compared to the simpler Toom-Cook method?

6.1 Winograd algorithm

A convolution can be expressed as a result of polynomials multiplication (see Chapter 3). Mapping the elements of a kernel vector **w** and an input vector **x** to coefficients of polynomials $w(a)$ and $x(a)$ respectively, the elements of output vector **s** (convolution of **w** and **x**) are equal to the coefficients of polynomial $s(a) = w(a)x(a)$. The Winograd family of algorithms for convolution is based on Chinese Remainder Theorem (CRT) for polynomials (Theorem 2).

The CRT says that for a polynomial $M(a)$ in ring of polynomials over a field $\mathbb{F}$, $M(a) = m_1(a) \dots m_\ell(a)$ where $m_i(a)$ for $i = 1, \dots, \ell$ are irreducible and pairwise coprime there exists $s(a)$ such as $\deg(s(a)) < \deg(M(a))$ the unique solution of system of congruences:

$$s(a) = s_i(a) \mod m_i(a)$$

and

$$s(a) = \sum_i s_i(a) N_i(a) M_i(a) \mod M(a) \tag{6.1}$$

where $N_i(a)M_i(a) + n_i(a)m_i(a) = 1$ and $M_i(a) = M(a)/m_i(a)$.

To compute the result of the convolution - the coefficients of the product of polynomials $w(a)$ and $x(a)$ - we put

$$\begin{aligned} s_i(a) &= w_i(a)x_i(a) \mod m_i(a), \\ w_i(a) &= w(a) \mod m_i(a), \\ x_i &= x(a) \mod m_i(a) \end{aligned}$$

Operations modulo $m_i(a)$ are equivalent to finding the remainder from division by $m_i(a)$; so if we assume that all polynomials $m_i(a)$ are of the first degree then the results in modulo $m_i(a)$ arithmetic are all constant polynomials (scalars)

$$w_i(a) = w(a) \mod m_i(a) = r_w$$

$$x_i(a) = x(a) \mod m_i(a) = r_x$$

Then we can perform the computations of $s_i(a) = w_i(a)x_i(a) \mod m_i(a)$ for $i = 1, \ldots, \ell$ as single multiplication

$$s_i(a) = r_w r_x$$

These operations for all $i = 1, \ldots, \ell$ are represented by Hadamard product of two vectors consisting of elements $w_1(a), \ldots, w_\ell(a)$ and $x_1(a), \ldots, x_\ell(a)$ (see Figure 6-1).

As described in Section 3.1 the following computations formula expresses two-dimensional Winograd convolution algorithm

$$\mathbf{A}^T(\mathbf{G}\mathbf{W}\mathbf{G}^T \odot \mathbf{B}^T\mathbf{X}\mathbf{B})\mathbf{A}$$

where matrices $\mathbf{W}$ and $\mathbf{X}$ represent kernel and input values, respectively.

If we use polynomial $m_i(a)$ of degree $d > 1$, then the results of $w_i(a) = w(a) \mod m_i(a)$ and $x_i(a) = x(a) \mod m_i(a)$ are polynomials, not scalars (see Figure 6-2). Thus, to compute $s_i(a)$ we need to multiply two polynomials $w_i(a)$, $x_i(a)$ rather than using simple scalar multiplication. However, to solve this sub-problem (i.e. computing the coefficients of the product of two polynomials $w_i(a)$ and $x_i(a)$) we can apply any suitable algorithm, including the Toom-Cook algorithm.

All polynomials $m_i(a)$ used in the Winograd algorithm have to be pairwise coprime, and similarly all polynomials used in the Toom-Cook algorithm to solve the sub-problem also need to be pairwise coprime. But polynomials in the two different groups do not need to be coprime. Some root points of poly-

nomials $m_i(a)$ such as 0, -1 and 1 (polynomials a, $a+1$ and $a-1$), offer superior floating point accuracy. This means that we can use them to solve both, an original and a sub-problem. In Figure 6-2 we can have $q_i = p_j$.

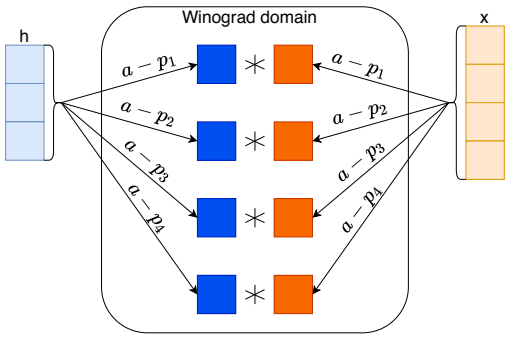


Figure 6-1: Transformation of the kernel (using matrix $\mathbf{G}$) and input (using matrix B^T) in one-dimensional Toom-Cook convolution algorithm $F_{(T-C)}(2,3)$ with four root points p_1, p_2, p_3 and p_4 (polynomials $a-p_1$, $a-p_2$, $a-p_3$ and $a-p_4$).

The approach with polynomials, $m_i(a)$ of degree $d > 1$ requires two steps of transformations (see Figure 6-2). Firstly, we transform input/kernel into the "Polynomials Winograd domain". This means we map input/kernel into polynomials of degree greater than zero. We then transform all polynomials into scalars in the "Winograd domain". To perform the second transformation we use the Toom-Cook algorithm. Similarly, after computing Hadamard product we first transform the result into "Polynomials Winograd domain" and consequently into the original domain. Each of these transformations can be represented by a matrix which multiplied by the input/kernel/output performs the transformation. We can merge the matrices for these two stages of transformation into a single transformation, allowing us to create three matrices $\mathbf{G^W}$, $\mathbf{B^W}$ and $\mathbf{A^W}$ applied to the kernel, input and result of the Hadamard product respectively. For the clarity, we denote matrices constructed for Toom-Cook algorithm as $\mathbf{G^{T-C}}$, $\mathbf{A^{T-C}}$ and $\mathbf{B^{T-C}}$. The exact method of constructing transformation matrices ($\mathbf{A^{T-C}}$, $\mathbf{B^{T-C}}$ and $\mathbf{G^{T-C}}$) is presented in Section 3.1.1 (Algorithm 6). Through the rest of this chapter we will also use the notation based on Winograd (Winograd 1980a) and follow by Lavin (Lavin and Gray 2016) where $F(m,k)$ denotes one-dimensional Winograd convolution algorithm with the output of the size m and the kernel of the size k. Respectively $F(m\times m, k\times k)$ de-

notes two-dimensional Winograd convolution algorithm with the output of the size $m \times m$ and the kernel of the size $k \times k$. When we consider only the most optimal version (Toom-Cook), we refer to it as $F_{T-C}(m, k)$ and $F_{T-C}(m \times m, k \times k)$ for one and two dimensions, respectively.

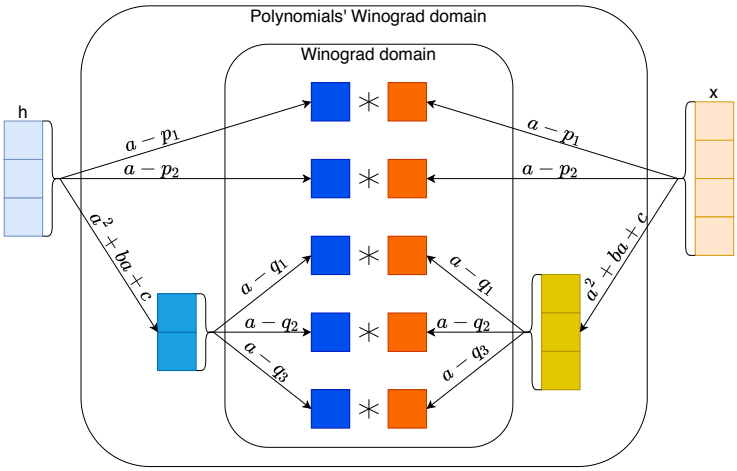


Figure 6-2: Transformation of kernel (matrix $\mathbf{G}$) and input (matrix $\mathbf{B}^T$) in one-dimensional Winograd convolution algorithm $F(2, 3)$ with polynomials $a - p_1$, $a - p_2$, $a^2 + ba + c$. The subproblem is solved with Toom-Cook algorithm $F_{T-C}(2, 2)$ for root points q_1, q_2, q_3 (polynomials $a - q_1$, $a - q_2$, $a - q_3$).

6.1.1 Matrices construction

Matrices $\mathbf{G^W}$ and $\mathbf{A^W}$

We use the function $\text{vec}(m(a))$ to map the polynomial $m(a) = m_1 + m_2 a + \ldots + m_n a^{n-1}$ to the vector

$$\text{vec}(m(a)) = \begin{bmatrix} m_1, & \ldots & , m_n \end{bmatrix}^T$$

We use $R_{m(a)}[p(a)]$ to denote the remainder of polynomial division of $p(a)$ by $m(a)$

$$R_{m(a)}[p(a)] = p(a) \mod m(a) \tag{6.2}$$

Matrices $\mathbf{G^W}$ and $\mathbf{A^W}$ are both constructed in the same way. They are also the same for one- and two-dimensional Winograd algorithm. Rows of the matrix $\mathbf{G^W}$ and $\mathbf{A^W}$, which stand for transformation with polynomials of the first

degree are identical to those in the Toom-Cook algorithm (see Algorithm 6) to within factors N_i). Then we have to construct the submatrices that correspond to the transformation with the polynomial $m_i(a)$ of the degree d higher than one. To do this we have to compose the matrix $\mathbf{G}$ with $\mathbf{G}'$ ($\mathbf{G^W} = \mathbf{G^{(T-C)}} \times \mathbf{G}'$), where $\mathbf{G}'$ represents transformation to the "Polynomials' Winograd domain" and the $\mathbf{G}$ matrix stands for transformation to the "Winograd domain" and is equal to matrix $\mathbf{G^{T-C}}$ of apropriate size (used in algorithm $F_{T-C}(d,d)$).

Similarly, matrix $\mathbf{A^W} = \mathbf{A^{T-C}} \times \mathbf{A}'$, where $\mathbf{A^{T-C}}$ is generated by the Toom-Cook algorithm $F_{T-C}(d,d)$ and $\mathbf{A}'$ stands for transformation into the "Polynomial Winograd domain" with polynomials $m_i(a)$ of the degree higher than 1. The last rows $\mathbf{A}_\ell$ and $\mathbf{G}_\ell$ represent the pseudo point ∞ needed to construct the modified version of the algorithm.

Below we present an example of the construction of matrices $\mathbf{A^W}$ and $\mathbf{G^W}$ for kernel of size $3/3 \times 3$ and output of size $2/2 \times 2$, choosing polynomials $m_1(a) = a$ and $m_2(a) = a^2 + ba + c$. To solve the sub-problem we use Toom-Cook algorithm $F_{T-C}(2,2)/F_{T-C}(2\times 2, 2\times 2)$ with root points 0, 1 (polynomials a and $a-1$).

$$\mathbf{G_1} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{A_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{G_2} = \mathbf{G^{T-C}} \times \mathbf{G}' = \begin{bmatrix} -1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c \\ 0 & 1 & -b \end{bmatrix} = \begin{bmatrix} -1 & 0 & c \\ 1 & 1 & -b-c \\ 0 & 1 & -b \end{bmatrix}$$

$$\mathbf{A_2} = \mathbf{A^{T-C}} \times \mathbf{A}' = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c & bc \\ 0 & 1 & -b & b^2-c \end{bmatrix} = \begin{bmatrix} 1 & 0 & -c & bc \\ 1 & 1 & -b-c & bc+b^2-c \\ 0 & 1 & -b & b^2-c \end{bmatrix}$$

$$\mathbf{G^W} = \begin{bmatrix} & \mathbf{G_1} & \\ & \mathbf{G_2} & \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & c \\ 1 & 1 & -b-c \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A^W} = \begin{bmatrix} & & \mathbf{A_1} & \\ & & \mathbf{A_2} & \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & -c & bc \\ 1 & 1 & -b-c & bc+b^2-c \\ 0 & 1 & -b & b^2-c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The exact algorithms to compute matrices $\mathbf{G^W}$ and $\mathbf{A^W}$ are presented in algorithm (7).

Matrix $\mathbf{B^W}$

First we construct auxiliary matrix $\mathbf{C}$ that includes blocks $\mathbf{C_i}$ for $i = 1, \cdots, \ell$ where ℓ is the number of the polynomials $m_i(a)$. The matrix $\mathbf{C}$ represents transformation from the "Polynomial Winograd domain" into the "Winograd domain".

The rows which stand for transformation with polynomials $m_i(a)$ of the first degree are equal to identity matrix. Blocks which stand for transformation with polynomial $m_i(a)$ of degree d greater than 1 represents transformation with matrix $\mathbf{B^{T-C}}$, generated for sub-problem with $F_{T-C}(d, d)$. A second matrix $\mathbf{E}$ includes the rest of operations, that is modulo $M(a)$ (remainder) from product of polynomials $M_i(a)$ and the polynomial obtained from extended Euclidean algorithm $N_i(a)$ (see Equation (6.1)). Additional zeros in rows of matrix $\mathbf{E}$ and column with coefficients of the polynomial $M_i(a)$ implement the modified version of the Winograd algorithm. The exact algorithm to compute matrix $\mathbf{B^W}$ is presented in Algorithm 8.

ALGORITHM 7: Construction of matrix $\mathbf{G^W}$ and $\mathbf{A^W}$ to transform kernel and result of Hadamard product in a Winograd convolution algorithm $F(m, k)$

Input: m - size of output,

k - size of kernel,

$\{m_1(a), \cdots, m_\ell(a)\}$ set of ℓ irreducible and pairwise coprime polynomials such as $\sum_i \deg(m_i(a)) = k + m - 2$

Output: Matrices $\mathbf{G^W}$ and $\mathbf{A^W}$ for Winograd convolution

$n = k + m - 2$

for $i = 1$ *to* ℓ **do**

$d = \deg(m_i(a))$

if $d == 1$ **then**

$p_i = root(m_i(a))$

$\mathbf{G_i} = \left[p_i^0, \ldots, p_i^{k-1}\right]$,

$\mathbf{A_i} = \left[p_i^0, \ldots, p_i^{m-1}\right]$

end

if $d > 1$ **then**

$\mathbf{G^{T-C}}$ - matrix $\mathbf{G}$ for $F_{(T-C)}(d, d)$

$\mathbf{G'} = \left[\text{vec}(R_{m_i(a)}[a^0]), \ldots, \text{vec}(R_{m_i(a)}[a^{k-1}])\right]$

$\mathbf{A^{T-C}}$ matrix $\mathbf{A}$ for $F_{(T-C)}(d, d)$

$\mathbf{A'} = \left[\text{vec}(R_{m_i(a)}[a^0]), \ldots, \text{vec}(R_{m_i(a)}[a^{m-1}])\right]$

$\mathbf{G_i} = \mathbf{G^{(T-C)}G'}$

$\mathbf{A_i} = \mathbf{A^{(T-C)}A'}$

end

$\mathbf{G_{\ell+1}} = [0, \ldots, 0, 1]$

$\mathbf{A_{\ell+1}} = [0, \ldots, 0, 1]$

end

$$\mathbf{G^W} = \begin{bmatrix} \mathbf{G_1} \\ \vdots \\ \mathbf{G_\ell} \\ \mathbf{G_{\ell+1}} \end{bmatrix} \quad \mathbf{A^W} = \begin{bmatrix} \mathbf{A_1} \\ \vdots \\ \mathbf{A_\ell} \\ \mathbf{A_{\ell+1}} \end{bmatrix}$$

ALGORITHM 8: Construction of matrix B to transform the input in the Winograd convolution algorithm $F(m, k)$

Input: m - size of output,

k - size of kernel

$\{m_1(a), \cdots, m_{\ell(a)}\}$ set of ℓ irredicible and pairwise coprime polynomials such as $\sum_i \deg(m_i(a)) = k + m - 2$

Output: Matrix B^W for Winograd convolution

$n = k + m - 2$

$M(a) = \prod_i m_i(a)$

$M_i(a) = M(a)/m_i(a)$

for $i = 1$ *to* ℓ **do**

$d = \deg(m_i(a))$

if $d == 1$ **then**

$\mathbf{C_i} = [1]$

end

if $d > 1$ **then**

$\mathbf{B^{T-C}}$ matrix $\mathbf{B}$ for $F_{T-C}(d, d)$

for $j = 1$ *to* $2d - 1$ **do**

$b_j(a) = \mathbf{B^{T-C}_{1,j}} + \mathbf{B^{T-C}_{2,j}} a + \ldots + \mathbf{B^{T-C}_{2d-1,j}} a^{2d-2}$

end

$\mathbf{C_i} = \begin{bmatrix} \text{vec}(R_{m_i(a)}[b_1(a)], & \ldots, & \text{vec}(R_{m_i(a)}[b_{2d-1}(a)]) \end{bmatrix}$

end

end

$$\mathbf{C} = \begin{bmatrix} \mathbf{C_1} & & \\ & \ddots & \\ & & \mathbf{C}_\ell \end{bmatrix}$$

for $i = 1$ *to* ℓ **do**

$N_i(a)$ — polynomial obtained from extended Euclidean algorithm for polynomials $m_i(a)$ and $M_i(a)$

$\mathbf{E_i} = \begin{bmatrix} \text{vec}(R_{M(a)}[a^0 N_i(a) M_i(a)]), \ldots, \text{vec}(R_{M(a)}[a^{d-1} N_i(a) M_i(a)]) \end{bmatrix}$

end

$$\mathbf{E} = \begin{bmatrix} \mathbf{E_1} & \cdots & \mathbf{E}_\ell \\ 0 & \cdots & 0 \end{bmatrix}$$

$$\mathbf{B^W} = \begin{bmatrix} \mathbf{E} \times \mathbf{C} & \text{vec}(M(a)) \end{bmatrix}$$

We present an example of constructing matrix $\mathbf{B^W}$ for kernels of the size $3/3 \times 3$ and outputs of size $2/2 \times 2$, chosing polynomials: $m_1(a) = a$ and $m_2(a) = a^2 + ba + c$ (as in Section 6.1.1). Matrix $\mathbf{B^{(T-C)}}$ is generated by the Toom-Cook algorithm $F_{T-C}(2 \times 2, 2 \times 2)$ with root points $0, 1$.

$$\mathbf{B^{(T-C)}} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{C_2} = \begin{bmatrix} -1 & 0 & -c \\ 1 & 1 & -b-1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -c \\ 0 & 1 & 1 & -b-1 \end{bmatrix}$$

Next, we construct the blocks of matrix $\mathbf{E}$. The polynomials obtained from extended Euclidean algorithm (Biggs 2002) are: $N_1 = 1$, $N_2 = -a$.

$$\mathbf{E_1} = \begin{bmatrix} 1 \\ b \\ c \end{bmatrix} \qquad \mathbf{E_2} = \begin{bmatrix} 0 & 0 \\ 0 & c \\ -1 & b \end{bmatrix} \qquad \mathbf{E} = \begin{bmatrix} 1 & 0 & 0 \\ b & 0 & c \\ c & -1 & b \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{E} \times \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ b & 0 & c \\ c & -1 & b \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -c \\ 0 & 1 & 1 & -b-1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ b & c & c & -c(b+1) \\ c & b+1 & b & c-b(b+1) \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B^W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ b & c & c & -c(b+1) & c \\ c & b+1 & b & c-b(b+1) & b \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

6.2 Optimality of Winograd algorithm

The family of Toom-Cook convolution algorithms have an optimal number of multiplications $n = (k + m - 1)^2$ (for two dimensions) for fixed k and m. While

computing convolution in DNNs, we break our input into the pieces of the size equal to algorithm input tile. This results in overlap of input tiles at boundaries. The exact number of overlapping input values for whole input depends on the kernel and input/output sizes (see description in (Lavin and Gray 2016)). We express the performance of the algorithm as the ratio of the number of multiplications per single output value and we denote as $ratio$. Thus, Toom-Cook algorithm $F_{T-C}(2 \times 2, 3 \times 3)$ requires 16 multiplications to compute 4 output values, so we have ratio equal to 4. For algorithm $F_{T-C}(4 \times 4, 3 \times 3)$, the $ratio = 2.25$. For Toom-Cook convolution with a fixed kernel size, the $ratio$ decreases with tile size. The bigger input/output tile, the fewer general multiplications are needed (see Table 5.2). The element-wise multiplications dominate the execution time of DNN convolution, so reducing number of general multiplications, reduces execution time. Unfortunately, with increasing the input/output size the floating point error of the computations increases exponentially (see Chapter 4).

When we apply Toom-Cook algorithm $F_{T-C}(m \times m, k \times k)$ the $ratio$ is equal to $(k + m - 1)^2/m^2$. In the Winograd method, as we can see from matrix construction, introducing polynomials $m_i(a)$ of the degree greater than one results in larger matrix sizes, which means the bigger (not optimal) number of general multiplications. Every Toom-Cook algorithm $F_{T-C}(d \times d, d \times d)$ used to solve sub-problem in Winograd algorithm requires $2d - 1$ polynomials of the first degree. The bigger number and higher degree polynomials we use the more additional general multiplications per output value are required. To compute $F(2 \times 2, 3 \times 3)$ we can use

- Four polynomials of the first degree (Toom-Cook algorithm). It requires 16 general multiplications to compute 4 output values, so $ratio = 16/4 = 4$.

- Two polynomials of the first degree and one of the second degree. It requires 25 general multiplications to compute 4 output values, so $ratio = (2 + 3)^2/4 = 6.25$.

- One polynomial of the first degree and one of the third degree. It requires 36 general multiplications to compute 4 output values, so $ratio = (1+5)^2/4 = 9$.

- Two polynomials of the second degree. It requires 36 general multiplications to compute 4 output values, so $ratio = (3+3)^2/4 = 9$.

- One polynomial of the fourth degree. It requires 49 general multiplications to compute 4 output values, so $ratio = 7^2/4 = 12.25$.

We can notice that in the above example using the polynomial $m_i(a)$ of the 4th degree does not change an input (mapped to the polynomial of the 3rd degree) and a kernel (mapped to the polynomial of the 2nd degree) pending transformations, so in this case it only introduces additional multiplications into convolution computations. Analogously using polynomial $m_i(a)$ of the 3rd degree does not change the kernel. For this reason we will not consider such cases in presented work.

Table 6.1: Number of multiplications for single output value in two-dimensional Winograd convolution algorithm for kernel 3×3 and outputs: 2×2, 4×4 and 6×6, for various number of the polynomials of the first and second degree used in CRT. In orange is Toom-Cook algorithm with all polynomials of the first degree.

	2×2			4×4				6×6				
$\deg = 1$	4	2	0	6	4	2	0	8	6	4	2	0
$\deg = 2$	0	1	2	0	1	2	3	0	1	2	3	4
Ratio	4	6.25	9	2.25	3.06	4	5.06	1.78	2.25	2.78	3.36	4

The Winograd method allows us to construct algorithms with different $ratio$s for fixed kernel and output size, while the Toom-Cook method has a constant $ratio$ for given k and m. Thus, for fixed kernel size we can construct sets of Winograd matrices with the same $ratio$ but different output/input size. For example for $F_{T-C}(4 \times 4, 3 \times 3)$, $ratio = 36/16 = 2.25$ and $F_W(6 \times 6, 3 \times 3)$, with 6 polynomials of the first degree, and one polynomial of the second degree, we have the same $ratio = 81/36 = 2.25$ – see Table 6.1. Given these choices with the

same computational $ratio$, we can investigate the floating point error of such algorithms and pick the more accurate one.

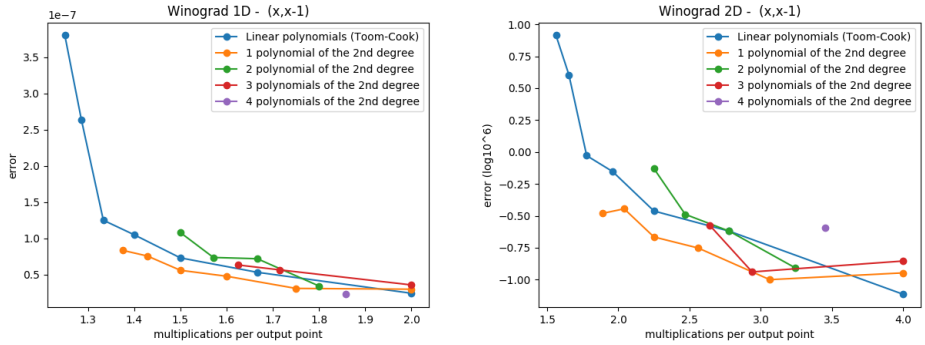
6.3 Tests results

6.3.1 Random data

We tested the accuracy of the Winograd convolution algorithm for the kernel of the size 3(1D) and 3×3(2D). We studied a range of output tile sizes from 2 to 8 (1D) and from 2×2 to 8×8 (2D). We run our initial experiments over 5000 loops where kernel and input values were sampled randomly from normal distribution with mean equal to 0 and standard deviation equal to 1. We computed the Euclidean error of Winograd convolution performed in $float32$ and compared it with the direct convolution in $float64$.

We investigated Winograd convolution algorithm with the most promising configurations of polynomials of the first and second degree (using the kernel of size 3 or 3×3). The best results for each $ratio$ (number of general multiplications per output value) and polynomial degree configuration are presented in Figure 6-3. We construct the first degree polynomials using known good root points: 0, -1, 1, $-1/2$, 2, $1/2$, -2, $-1/4$, 4 (see Table 5.6). As second degree polynomials, we pick those with the coefficients equal to 0, -1 and 1, coprime with the polynomials of the first degree. That is: a^2+1, a^2+a+1 and a^2-a+1. To solve the sub-problem for the polynomial of degree greater than 1, we use Toom-Cook convolution algorithm $F_{T-C}(2 \times 2, 2 \times 2)$ and root points 0, 1 and ∞.

We can notice that the accuracy of Winograd convolution does not always decrease smoothly with the $ratio$, at least with our way of choosing polynomials. It is the result of matrix properties where the symmetric root points a,$-a$,$1/a$,$-1/a$ work best. For example, Winograd with output 8×8 requires ten root points for the Toom-Cook version (all polynomials $m_i(a)$ of the first degree). Replacing two of them by the polynomial of the degree $d = 2$ results in eight symmetric root points 0, $-1/2$, $1/2$, -1, 1, -2, 2 and ∞ and the poly-



(a) (b)

Figure 6-3: Euclidean error of Winograd convolution in $float32$ comparing to the direct method computed in $float64$

nomial $a^2 + 1$. This configuration is good for accuracy. When increasing the number of root points to 11 we have to introduce one more root point: $-1/4$ which has a great impact on the floating point error. When we use only one polynomial of the second degree, we found that $a^2 + 1$ works best. It provides only two non zero coefficients instead of three (for $a^2 + a + 1$ and $a^2 - a + 1$).

We can also observe that for smaller $ratio$s (up to $ratio$ around 1.9 and 3.5 for one- and two-dimensional convolution) – that means lower time complexity – the Winograd algorithm with one polynomial of the second degree gives smaller floating point error. For $ratio$ equal to 1.9 Winograd algorithm reduces the error by over 20% and for $ratio$ equal to 2.25 nearly 40% compared to Toom-Cook, for one- and two-dimensional convolution respectively (see Figure 6-3). Thus, we are able to decrease the error keeping the same $ratio$ or improve time performance keeping similar floating point error of computations.

6.3.2 Extended example

We run simple experiments for the Vgg16 CNN using Tensorflow Slim (Abadi et al. 2015; Sergio Guadarrama 2016) with thirteen convolution layers, with the kernel of the size 3×3. As inputs we used 2000 images from the ImageNet validation set (Deng et al. 2009). The computations were done in $float32$ and

intermediate results were casted to half precision numbers system $float16$ and $bfloat16$.

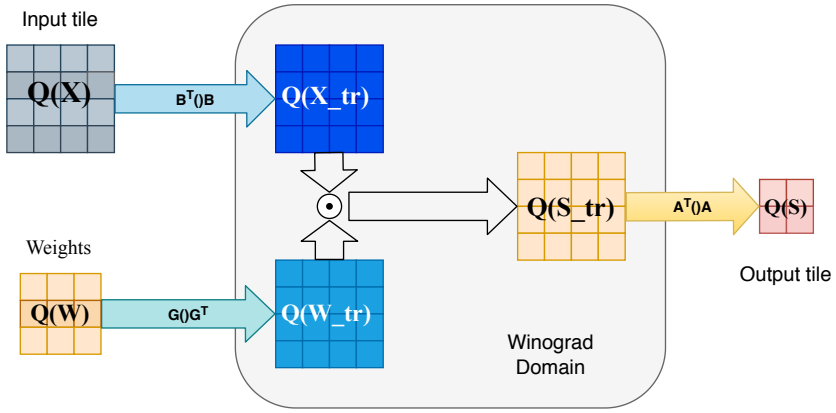


Figure 6-4: Winograd convolution overview - cast

We tested the Toom-Cook algorithms with outputs 4×4, 6×6 and 8×8. This means the $ratio$ of multiplications per single ouptut point was 2.25, 1.78 and 1.56 respectively (see Table 6.1). For comparison we chose the Winograd algorithm with one polynomial of the second degree for even output sizes, from 6×6 up to 12×12. The $ratio$ of multiplications per single output value was 2.25, 1.89, 1.69 and 1.56 respectively.

In our initial tests on random data, we have found that using $a^2 + 1$ as the polynomial of the second degree works best. Polynomials of the first degree for $F_W(m \times m, 3 \times 3)$ were identical as those used for $F_{T-C}((m-2) \times (m-2), 3 \times 3)$ Table 5.5.

In our tests, we used Winograd algorithms with only one polynomial of the second degree. We could achieve better accuracy by using more second degree polynomials, but this would be at the cost of a worse computational $ratio$. We focused on the cases where the image recognition accuracy decreases – $ratio$ equal to 2.25 in $float16$ and $ratio$ between 1.78 and 1.56 in $bfloat16$. We do not present the all possible results, e.g. for $F_W(12 \times 12, 3 \times 3)$ with 2 polynomials of the second degree ($ratio = 1.78$), but our results are indicative.

We looked at the percentage of image recognition ($top - 1$) for Vgg16 network with Winograd convolution layers in comparison to the same network

Table 6.2: Percentage of image recognition for Toom-Cook convolution algorithm for kernel of the size 3×3 and outputs 4×4, 6×6 and 8×8 in $float32$, $float16$ and $bfloat16$

	dir	T-C(4×4)	T-C(6×6)	T-C(8×8)
ratio	9	2.25	1.78	1.56
$float32$	70%	70%	70%	70%
$float16$	70%	10%	0.05%	0.05%
$bfloat16$	70%	70%	70%	68%

Table 6.3: Percentage of image recognition for Winograd convolution algorithm with one polynomial of the second degree a^2+1 for kernel of the size 3×3 and outputs 6×6, 8×8, 10×10 and 12×12 in $float32$, $float16$ and $bfloat16$.

	dir	W(6×6)	W(8×8)	W(10×10)	W(12×12)
ratio	9	2.25	1.89	1.69	1.56
$float32$	70%	70%	70%	70%	70%
$float16$	70%	65%	0.1%	0.05%	0.05%
$bfloat16$	70%	70%	70%	70%	62%

with direct convolution using the same floating point precision. In Table 6.2 and Table 6.3 we present the results for various floating point precisions (see Section 2.3). For the output sizes under consideration, we do not observe any changes in network accuracy when using $float32$. In $float16$, all investigated versions of Toom-Cook algorithms failed due to under/overflows. In $bfloat16$ the percentage of image recognition for Toom-Cook algorithm with output 6×6 ($ratio$ equal to 1.78) is the same as for direct convolution , but for output size 8×8 the accuracy decreases.

With $float16$, we see that using Winograd convolution instead of Toom-Cook with the same performance $ratio$ (equal to 2.25), increases the recognition accuracy from 10% to 65%. The main problem we face with $float16$ is that it cannot store the same range of values as $float32$. Then using the same good root points (like 0, -1 and 1) more than once results in lower intermediate values, and less likelihood of overflow.

Using $bfloat16$, the decrease in image recognition appears for bigger input sizes than in $float16$. The $bfloat16$ format allows us to represent nearly the same range of values as single precision (see Section 2.3). However, the lower number of bits results in lower accuracy of values representation and larger floating point error from operations. In our experiments we can observe the impact of this for network with Toom-Cook convolution algorithm with output of the size 8×8. We have not found a configuration of polynomials that would give us the accuracy of image recognition better than 68% with the $ratio$ equal to 1.56. We construct the Winograd algorithm with the accuracy of image recognition equal to 70% (the same accuracy we get using of a direct convolution algorithm) with $ratio = 1.69$. Presented results are only an example how Winograd convolution algorithm might be beneficial in improving the accuracy of computations in DNNs.

6.4 Conclusions

In this chapter we describe and provide the explicit algorithm for the construction of Winograd transformation matrices in a general case. We show that the main benefit of using superlinear polynomials is that the same good root points can be used multiple times, which improves floating point accuracy. The Toom-Cook method allows to find a balance between number of general multiplications and floating points accuracy by varying the tile size. The presented Winograd method offers a larger space of trade-offs between computation complexity and accuracy using higher order polynomials. Thus, it allows us to find an attractive solutions that are not available when using Toom-Cook. The similar approach is presented in (Meng and Brothers 2019). However, they used one superlinear polynomial and do not provided any general method of constructing transformation matrices .

In presented case study we find that in $bfloat16$ precision we can construct an algorithm that maintains the same accuracy of image recognition as Toom-Cook, but has better $ratio$ of elementwise multiplications per single output

value than Toom-Cook. In $float16$ precision we can obtain better accuracy using Winograd convolution algorithm with one polynomial of the second degree, as compared to Toom-Cook (for the kernel of the size 3×3, and output 4×4) with the same $ratio$ of number of elementwise multiplications per output value. The presented Winograd convolution algorithm does not require additional operations in the transformation to/from the "Winograd domain", and although the Winograd method itself is complex, the generated convolution algorithm does not require a more advanced implementation.

Chapter 7

Discussion and Conclusions

This thesis contains our results achieved studying the problem of numerical errors which occur during calculations of the Winograd fast convolution algorithms. These algorithms are widely used in many problems for which Deep Neural Networks are applied. We achieved state-of-the-art results developing a set of novel methods that significantly improve the accuracy of the computational results, not increasing their complexity (in terms of the number of operations). Our work integrates several various approaches to the problem.

Four major contributions of our thesis are detailed below.

Firstly, after investigation a mathematical background and providing the exact algorithms with pseudocodes for a construction of transformation matrices (Chapter 3) we fixed an idea which parameters of Toom-Cook/Winograd convolution algorithms we can manipulate. We had found that it is advisable to tune not only kernel and output sizes but also the number, degree and root points of polynomials used to construct transformation matrices. As a matter of the fact, it appears that the family of the Winograd convolution algorithms is wider than practically used in machine learning field. We investigated in more details the simpler, most optimal subclass of it, that is Toom-Cook algorithms.

Secondly, we identified in a theoretical way the components of numerical errors of the Toom-Cook convolution algorithms (Chapter 4). We proved that the accuracy of the computational results of these algorithms depends on transformation matrices, on the summation method in dot product computations,

the method of summation over input channels and the precision of the number system in which we perform computations. We proved that the errors of the convolution computations grows exponentially with the output sizes. We also demonstrated that the errors of the modified Toom-Cook algorithm is smaller than the errors of the Toom-Cook algorithm, although still grows exponentially.

Thirdly, we proposed a couple of techniques that improve the accuracy of the Toom-Cook convolution computations based on our theoretical results and verified them empirically for the wide range of input sizes and the most commonly used kernel sizes (Chapter 5). We investigated in details, the way of choosing points which determines properties of the transformation matrices. We constructed sets of optimal points for various input tile sizes, both in a single and a mixed precision. We proposed and tested the canonical summation order, based on the idea of the Huffman coding, to reduce further the error in dot product computations. In addition we used the pairwise summation over input channels instead of a linear one.

All the proposed methods give an error reduction of the floating point computation up to 50% over the state-of-the-art.

Finally, we analyzed the Winograd algorithms (Chapter 6). Despite the fact that these algorithms are not optimal in terms of the number of general multiplications, we show that we can reduce this number in context of the whole layer by increasing output size. Based on our performance analysis and the fact that one particular Winograd convolution algorithm has been successfully applied by Meng and Brothers (Meng and Brothers 2019) we come to the conclusion that it would be worth to investigate this technique further, particularly for lower precisions and bigger output sizes.

Our work has been done in the general context and tested mainly on random data. The actual results of how much we can speed up the convolution computations depends for sure on the particular network architecture, computer architecture, processor and input data properties like for example an image size if we consider pattern recognition problems.

It can also be interesting to investigate much bigger input sizes, for example 30×30. For such big matrices the dependency between the errors from ill-conditioning can have even bigger impact on the accuracy. Techniques like Chebyshev nodes or matrix factorizations can appear to be extremely useful in this case.

Appendix A

Norm Equivalence - Some Inequalities

In Chapter 2, Section 2.2.4 we introduced definitions of vector and matrix norms in vector spaces $\mathbb{R}^n$ or $\mathbb{R}^{m\times n}$, respectively. In Chapter 4, Section 4.3 we also proved the following theorem.

Theorem 10. *Any two norms defined on the same vector space of finite dimension are equivalent.*

Now we demonstrate some norm inequalities which are introduced in Chapter 4 and used throughout the thesis.

For any vector $\mathbf{x} \in \mathbb{R}^n$ it holds

$$\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2 \tag{A.1}$$

For any matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ we have

$$\frac{1}{\sqrt{m}}\|\mathbf{W}\|_1 \leq \|\mathbf{W}\|_2 \leq \sqrt{n}\|\mathbf{W}\|_1 \tag{A.2}$$

$$\|\mathbf{W}\|_2 \leq \|\mathbf{W}\|_F \leq \sqrt{min(m,n)}\|\mathbf{W}\|_2 \tag{A.3}$$

Proof. Based on (Rudin 1986)

To justify the first inequality of Equation (A.1), let us observe that

$$\|\mathbf{x}\|_2^2 = \sum_i x_i^2 \leq \sum_i |x_i| \cdot \sum_i |x_i| = \|\mathbf{x}\|_1^2$$

Hence $\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1$.

For the second inequality of Equation (A.2) let us notice

$$\|\mathbf{x}\|_1 = \sum_i |x_i| =< \mathbf{1}, |\mathbf{x}| >= \sum_i 1 \cdot |x_i| \leq$$

continuing, from the Buniakowski-Schwartz inequality (see Section 2.2.5) we obtain

$$\leq \sqrt{< \mathbf{1}, \mathbf{1} >} \cdot \sqrt{< \mathbf{x}, \mathbf{x} >} = \sqrt{\sum_i 1} \cdot \sqrt{\sum_i x_i^2} = \sqrt{n}\|\mathbf{x}\|_2$$

what gives $\|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2$ and finishes the proof of the above inequalities Equation (A.1) for vector norms.

Given a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ we have

$$\|\mathbf{W}\|_2 = \max_{\mathbf{x} \neq 0} \frac{\|\mathbf{W}\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\mathbf{x} \neq 0} \|\mathbf{W} \frac{\mathbf{x}}{\|\mathbf{x}\|}\|$$

what implies, that

$$\|\mathbf{W}\|_2 = \max_{\|\mathbf{x}\|_2 = 1} \|\mathbf{W} \cdot \mathbf{x}\|$$

Now from inequalities (1) and norm properties we obtain

$$\|\mathbf{W}\mathbf{x}\|_2 \leq \|\mathbf{W}\mathbf{x}\|_1 \leq \|\mathbf{W}\|_1 \|\mathbf{x}\|_1 \leq \|\mathbf{W}\|_1 \sqrt{n} \|\mathbf{x}\|_2$$

Taking a maximum of both sides for $\mathbf{x} \in \mathbb{R}^n$ such that $\|\mathbf{x}\|_2 = 1$ we obtain finally

$$\|\mathbf{W}\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{W}\|_1$$

To obtain the second inequality of Equation (A.2) notice that for a vector $\mathbf{x} \in \mathbb{R}^m$ we have $\|\mathbf{Wx}\|_1 \leq \sqrt{m}\|\mathbf{Wx}\|_2$ from (1).
Then $\sqrt{m}\|\mathbf{Wx}\|_2 \leq \sqrt{m}\|\mathbf{W}\|_2\|\mathbf{x}\|_2 \leq \sqrt{m}\|\mathbf{W}\|_2\|\mathbf{x}\|_1$.
Hence

$$(**) \quad \|\mathbf{Wx}\|_1 \leq \sqrt{m}\|\mathbf{W}\|_2\|\mathbf{x}\|_1$$

Remind that 1-norm of a matrix $\|\mathbf{W}\|_1$ it is a maximal value of the sums of entries of columns. Let us assume that the maximum is achieved for the j-th column. That is $\|\mathbf{W}\|_1 = \sum_i |w_{ij}|$. Denote $\mathbf{e_j} = (0, ...0, 1, 0, ...0)$ a vector in $\mathbb{R}^n$. Thus we have

$$\|\mathbf{We_j}\|_1 = \|(w_{1j}, ..., w_{mj})\|_1 = \sum_i |w_{ij}| = \|\mathbf{W}\|_1$$

Now putting in $(**)$ $\|\mathbf{e_j}\| = 1$ we obtain

$$\|\mathbf{W}\|_1 \leq \sqrt{m}\|\mathbf{W}\|_2$$

So the inequalities of Equation (A.2) are proven.

To justify inequalities of Equation (A.3) let $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_2 = 1$ and $\mathbf{W} \in \mathbb{R}^{m\times n}$. Thus

$$\|\mathbf{Wx}\|_2 = \sum_i |\sum_j w_{ij}x_j|^2 \leq \sum_i (|\sum_j |w_{ij}||x_j|)^2 \leq$$

$$\leq \sum_i (|\sum_j |w_{ij}|^2)(\sum_j (|x_j|)^2) = \|\mathbf{W}\|_F^2$$

Hence

$$\|\mathbf{W}\|_2 \leq \|\mathbf{W}\|_F$$

and the first inequality of Equation (A.3) is justified. To prove the second inequality of Equation (A.3) observe that any matrix $\mathbf{W} \in \mathbb{R}^{m\times n}$ could be identified with a vector composed of its columns $\mathbf{W} = [\mathbf{W_1}, ..., \mathbf{W_n}]$ where $\mathbf{W_i} = [w_{1i}, ..., w_{mi}]^T$ is a vector equal to an i-th column of $\mathbf{W}$. This identification is in fact an isomorphism of the vector spaces $\mathbb{R}^{m\times n}$ and $(R^m)^n$. Such

isomorphic spaces have exactly the same properties, and identical norms, in particular.

Let us denote $\mathbf{e_j} = (0, ...0, 1, 0, ...0)$ a vector in $\mathbb{R}^m$ which all coefficients are equal 0 except on the j-th position which is equal 1. We have $\|\mathbf{e_j}\|_2 = 1$ and $\mathbf{We_j} = \mathbf{W_j}$. Hence $\|\mathbf{W}\|_2 \geq \|\mathbf{W}e_j\|_2 = \|\mathbf{W_j}\|_2$ and

$$n \cdot \|\mathbf{W}\|_2^2 \geq \sum_j \|\mathbf{W_j}\|_2^2.$$

Consequently

$$\|\mathbf{W}\|_2^2 \geq \frac{1}{n} \sum_j \|\mathbf{W_j}\|_2^2 = \frac{1}{n} \|\mathbf{W}\|_F^2$$

Thus $\|\mathbf{W}\|_F \leq \sqrt{n}\|\mathbf{W}\|_2$.

To finish the justification of Equation (A.3) observe that just from the definition we have $\|\mathbf{W}\|_F = \|\mathbf{W}^T\|_F$ and $\|\mathbf{W}\|_2 = \|\mathbf{W}^T\|_2$.
Hence from the last inequality it follows that $\|\mathbf{W}^T\|_F \leq \sqrt{m}\|\mathbf{W}^T\|_2$
and finally

$$\|\mathbf{W}\|_F \leq \sqrt{m}\|\mathbf{W}\|_2$$

Taking into account proved above inequality

$$\|\mathbf{W}\|_F \leq \sqrt{n}\|\mathbf{W}\|_2$$

we finish the proof of Equation (A.3). □

Appendix B

Toom-Cook Convolution Over Multiple Channels

B.1 Single precision - 1D

Table B.1: Toom-Cook over multiple channels in $float32$ - error per single output value for one-dimensional convolution.

m	1 channel	32 channels	64 channels
1	1.75E-08	2.74E-07	5.12E-07
2	2.45E-08	3.80E-07	7.03E-07
3	5.19E-08	7.08E-07	1.28E-06
4	6.92E-08	8.35E-07	1.48E-06
5	9.35E-08	1.09E-06	2.00E-06
6	1.15E-07	1.31E-06	2.34E-06
7	2.34E-07	2.90E-06	5.21E-06

Table B.2: Toom-Cook over 32 channels in $float32$ - error per single output value for one-dimensional convolution. Columns "ratio in %" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in %.

m	1 channel	linear summation	pairwise summation	ratio in %
1	1.75E-08	2.74E-07	1.90E-07	69%
2	2.45E-08	3.80E-07	2.71E-07	71%
3	5.19E-08	7.08E-07	5.11E-07	72%
4	6.92E-08	8.35E-07	6.17E-07	74%
5	9.35E-08	1.09E-06	8.35E-07	77%
6	1.15E-07	1.31E-06	9.79E-07	75%
7	2.34E-07	2.90E-06	2.16E-06	74%

Table B.3: Toom-Cook over 64 channels in $float32$ - error per single output value for one-dimensional convolution. Columns "ratio in %" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in %.

m	1 channel	linear summation	pairwise summation	Ratio in %
1	1.75E-08	5.12E-07	2.87E-07	56%
2	2.45E-08	7.03E-07	4.00E-07	57%
3	5.19E-08	1.28E-06	7.59E-07	59%
4	6.92E-08	1.48E-06	9.18E-07	62%
5	9.35E-08	2.00E-06	1.24E-06	62%
6	1.15E-07	2.34E-06	1.47E-06	63%
7	2.34E-07	5.21E-06	3.20E-06	61%

B.2 Single precision - 2D

Table B.4: Toom-Cook over multiple channels in $float32$ - error per single output value for two-dimensional convolution.

$m \times m$	1 channel	32 channels	64 channels
1×1	4.63E-08	5.25E-07	9.44E-07
2×2	7.65E-08	9.05E-07	1.65E-06
3×3	2.35E-07	2.87E-06	5.33E-06
4×4	3.29E-07	3.60E-06	6.56E-06
5×5	6.81E-07	7.78E-06	1.41E-05
6×6	8.79E-07	9.48E-06	1.71E-05
7×7	2.43E-06	4.66E-05	8.41E-05

Table B.5: Toom-Cook over 32 channels in $float32$ - error per single output value for two-dimensional convolution. Columns "ratio in $\%$" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in $\%$.

$m \times m$	1 channel	linear summation	pairwise summation	ratio in $\%$
1×1	4.63E-08	5.25E-07	3.95E-07	75%
2×2	7.65E-08	9.05E-07	6.47E-07	71%
3×3	2.35E-07	2.87E-06	2.09E-06	73%
4×4	3.29E-07	3.60E-06	2.70E-06	75%
5×5	6.81E-07	7.78E-06	5.71E-06	73%
6×6	8.79E-07	9.48E-06	7.12E-06	75%
7×7	2.43E-06	4.66E-05	3.41E-05	73%

Table B.6: Toom-Cook over 64 channels in $float32$ - error per single output value for two-dimensional convolution. Columns "ratio in $\%$" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in $\%$.

$m \times m$	1 channel	linear summation	pairwise summation	ratio in $\%$
1×1	4.63E-08	9.44E-07	5.83E-07	62$\%$
2×2	7.65E-08	1.65E-06	9.59E-07	58$\%$
3×3	2.35E-07	5.33E-06	3.11E-06	58$\%$
4×4	3.29E-07	6.56E-06	3.98E-06	61$\%$
5×5	6.81E-07	1.41E-05	8.57E-06	61$\%$
6×6	8.79E-07	1.71E-05	1.04E-05	61$\%$
7×7	2.43E-06	8.41E-05	5.09E-05	61$\%$

B.3 Mixed precision - 1D

Table B.7: Toom-Cook computation over multiple channels in $float32$ with transformations in $float64$ - error per single output value for one-dimensional convolution.

m	1 channel	32 channels	64 channels
1	1.75E-08	2.73E-07	5.15E-07
2	1.87E-08	3.60E-07	6.73E-07
3	3.66E-08	6.50E-07	1.20E-06
4	4.41E-08	7.45E-07	1.41E-06
5	6.09E-08	1.00E-06	1.92E-06
6	6.97E-08	1.17E-06	2.18E-06
7	1.55E-07	2.60E-06	4.91E-06

Table B.8: Toom-Cook computation over 32 channels in $float32$ with transformations in $float64$- error per single output value for one-dimensional convolution. Columns "ratio in $\%$" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in $\%$.

m	1 channel	linear summation	pairwise summation	ratio in $\%$
1	1.75E-08	2.73E-07	1.90E-07	70%
2	1.87E-08	3.60E-07	2.31E-07	64%
3	3.66E-08	6.50E-07	4.39E-07	68%
4	4.41E-08	7.45E-07	5.16E-07	69%
5	6.09E-08	1.00E-06	6.98E-07	70%
6	6.97E-08	1.17E-06	7.90E-07	68%
7	1.55E-07	2.60E-06	1.80E-06	69%

Table B.9: Toom-Cook computation over 64 channels in $float32$ with transformations in $float64$- error per single output value for one-dimensional convolution. Columns "ratio in $\%$" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in $\%$.

m	1 channel	linear summation	pairwise summation	
1	1.75E-08	5.15E-07	2.88E-07	56%
2	1.87E-08	6.73E-07	3.58E-07	53%
3	3.66E-08	1.20E-06	6.72E-07	56%
4	4.41E-08	1.41E-06	7.86E-07	56%
5	6.09E-08	1.92E-06	1.06E-06	55%
6	6.97E-08	2.18E-06	1.20E-06	55%
7	1.55E-07	4.91E-06	2.75E-06	56%

B.4 Mixed precision - 2D

Table B.10: Toom-Cook computation over multiple channels in $float32$ with transformations in $float64$ - error per single output value for two-dimensional convolution.

$m \times m$	1 channel	32 channels	64 channels
1×1	4.63E-08	5.25E-07	9.48E-07
2×2	5.27E-08	8.51E-07	1.59E-06
3×3	1.62E-07	2.70E-06	5.13E-06
4×4	2.14E-07	3.60E-06	6.68E-06
5×5	3.69E-07	6.06E-06	1.14E-05
6×6	5.18E-07	8.48E-06	1.59E-05
7×7	3.39E-06	4.21E-05	8.03E-05

Table B.11: Toom-Cook computation over 32 channels in $float32$ with transformations in $float64$- error per single output value for two-dimensional convolution. Columns "ratio in $\%$" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in $\%$.

$m \times m$	1 channel	linear summation	pairwise summation	ratio in $\%$
1×1	4.63E-08	5.25E-07	3.95E-07	75%
2×2	5.27E-08	8.51E-07	5.54E-07	65%
3×3	1.62E-07	2.70E-06	1.80E-06	67%
4×4	2.14E-07	3.60E-06	2.36E-06	66%
5×5	3.69E-07	6.06E-06	4.07E-06	67%
6×6	5.18E-07	8.48E-06	5.64E-06	67%
7×7	3.39E-06	4.21E-05	2.91E-05	69%

Table B.12: Toom-Cook computation over 64 channels in $float32$ with transformations in $float64$- error per single output value for two-dimensional convolution. Columns "ratio in $\%$" present the ratio of error per single output value achieved with pairwise summation and error per single output value with linear summation in $\%$.

$m \times m$	1 channel	linear summation	pairwise summation	ratio in $\%$
1×1	4.63E-08	9.48E-07	5.85E-07	62%
2×2	5.27E-08	1.59E-06	8.48E-07	53%
3×3	1.62E-07	5.13E-06	2.75E-06	54%
4×4	2.14E-07	6.68E-06	3.61E-06	54%
5×5	3.69E-07	1.14E-05	6.17E-06	54%
6×6	5.18E-07	1.59E-05	8.53E-06	54%
7×7	3.39E-06	8.03E-05	4.34E-05	54%

B.5 Summary

Table B.13: Ratio of Toom-Cook in $float32$ with linear summation over the channels and in mixed precision with pairwise summation over the channels - error for single output value in 1 and 2 dimensions.

1D			2D		
m	32 channels	64 channels	out size	32 channels	64 channels
1	69%	56%	1×1	75%	62%
2	61%	51%	2×2	61%	51%
3	62%	53%	3×3	63%	52%
4	62%	53%	4×4	66%	55%
5	65%	53%	5×5	52%	44%
6	60%	51%	6×6	59%	50%
7	62%	53%	7×7	62%	52%

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. URL: https://www.tensorflow.org/.

Alam, Syed Asad, James Garland, and David Gregg (2021). "Low-precision Logarithmic Number Systems". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 18, pp. 1 –25.

Alam, Syed Asad and David Gregg (2020). "Beyond Base-2 Logarithmic Number Systems". In: *Proceedings of the 21st ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES 2020, London, UK, June 16, 2020*. Ed. by Jingling Xue and Changhee Jung. ACM, pp. 141–145.

Anderson, Andrew, James Garland, Yuan Wen, Barbara Barabasz, Kaveena Persand, Aravind Vasudevan, and David Gregg (2019). *Hardware and software performance in deep learning*. URL: https://digital-library.theiet.org/content/books/10.1049/pbpc022e_ch6.

Antipov, Grigory, Sid-Ahmed Berrani, and Jean Luc Dugelay (Jan. 2016). "Minimalistic CNN-based ensemble model for gender prediction from face im-

ages". In: *Pattern Recognition Letters* 70, pp. 59–65. URL: https://hal.archives-ouvertes.fr/hal-01380573.

Ballard, Grey, Austin R. Benson, Alex Druinsky, Benjamin Lipshitz, and Oded Schwartz (2016). "Improving the numerical stability of fast matrix multiplication". In: *SIAM Journal on Matrix Analysis and Applications* 37.4, pp. 1382–1418.

Barabasz, Barbara, Andrew Anderson, Kirk M. Soodhalter, and David Gregg (2020). "Error Analysis and Improving the Accuracy of Winograd Convolution for Deep Neural Networks". In: *ACM Transactions on Mathematical Software (TOMS)* 46, pp. 1 –33.

Barabasz, Barbara and David Gregg (2019). "Winograd Convolution for DNNs: Beyond Linear Polynomials". In: *AI*IA 2019 - Advances in Artificial Intelligence - XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19-22, 2019, Proceedings*. Ed. by Mario Alviano, Gianluigi Greco, and Francesco Scarcello. Vol. 11946. Lecture Notes in Computer Science. Springer, pp. 307–320. URL: https://doi.org/10.1007/978-3-030-35166-3_22.

Biggs, Norman L. (2002). *Discrete Mathematics*. 2nd. New York, NY, USA: Oxford University Press.

Bini, Dario and Grazia Lotti (1980). "Stability of fast algorithms for matrix multiplication". In: *Numerische Mathematik* 36.1, pp. 63–72. URL: https://doi.org/10.1007/BF01395989.

Blahut, Richard E. (2010). *Fast Algorithms for Signal Processing*. New York, NY, USA: Cambridge University Press.

Bodrato, Marco (2007). "Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0". In: *Arithmetic of Finite Fields*. Ed. by Sunar B. Carlet C. Vol. 4547. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, pp. 116–133. URL: https://doi.org/10.1007/978-3-540-73074-3_10.

Bodrato, Marco and Alberto Zanoni (2006). *What about Toom-Cook Matrices Optimality ?* Tech. rep. 605. Centro Vito Volterra, Universita di Roma Tor Vergata.

— (2007). "Integer and Polynomial Multiplication: Towards Optimal Toom-Cook Matrices". In: *Proceedings of the ISSAC 2007 Conference*. Ed. by Christopher W. Brown. ACM press, pp. 17–24.

Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello (2016). "An Analysis of Deep Neural Network Models for Practical Applications". In: *CoRR* abs/1605.07678. URL: http://arxiv.org/abs/1605.07678.

Castaldo, Anthony M., R. Clint Whaley, and Anthony T. Chronopoulos (2008). "Reducing Floating Point Error in Dot Product Using the Superblock Family of Algorithms". In: *SIAM J. Scientific Computing* 31.2, pp. 1156–1174.

Chugh, Manik and Behrooz Parhami (2013). "Logarithmic arithmetic as an alternative to floating-point: A review". In: *2013 Asilomar Conference on Signals, Systems and Computers*. IEEE, pp. 1139–1143.

Cook, Stephen A. (1966). "On the Minimum Computation Time of Functions". PhD thesis. Cambridge, Mass.: Harvard University.

Cooley, James and John Tukey (1965). "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* 19.90, pp. 297–301.

Dahlqvist, Fredrik, Rocco Salvia, and George A. Constantinides (2019). "A Probabilistic Approach to Floating-Point Arithmetic". In: *53rd Asilomar Conference on Signals, Systems, and Computers, ACSCC 2019, Pacific Grove, CA, USA, November 3-6, 2019*. Ed. by Michael B. Matthews. IEEE, pp. 596–602. URL: https://doi.org/10.1109/IEEECONF44664.2019.9048893.

Datta, Biswa Nath (2010). *Numerical Linear Algebra and Applications, Second Edition*. 2nd. USA: Society for Industrial and Applied Mathematics.

Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng (2012). "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2012/file/\linebreak6aca97005c68f12068Paper.pdf.

Demmel, James (1997). *Applied Numerical Linear Algebra*. SIAM Publications. URL: http://bookstore.siam.org/ot56/.

Demmel, James, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg (2007). "Fast matrix multiplication is stable". In: *Numerische Mathematik* 106.2, pp. 199–224. URL: https://doi.org/10.1007/s00211-007-0061-6.

Demmel, James and Yozo Hida (2004). "Accurate and efficient floating point summation". In: *SIAM Journal on Scientific Computing* 25, pp. 1214–1248.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*.

Fernandez-Marques, Javier, Paul Whatmough, Andrew Mundy, and Matthew Mattina (2020). "Searching for Winograd-aware Quantized Networks". In: *Conference on Machine Learning and Systems (MLSys)*. URL: https://arxiv.org/pdf/2002.10711.pdf.

Fukushima, Kunihiko (1980). "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36, pp. 193–202.

Gautschi, Walter (1974). "Norm Estimates for Inverses of Vandermonde matrices". In: *Numerische Mathematik* 23.4, pp. 337–347. URL: https://doi.org/10.1007/BF01438260.

— (1990). "How (un)stable are Vandermonde systems?" In: *Asymptotic and Computational Analysis* 124, pp. 193–210.

— (2011). "Optimally scaled and optimally conditioned Vandermonde and Vandermonde-like matrices". In: *BIT Numerical Mathematics* 51.1, pp. 103–125. eprint: http://link.springer.com/content/pdf/10.1007%2Fs10543-010-0293-1.pdf.

Goldberg, David (1991). "What every computer scientist should know about floating-point arithmetic". In: *ACM Computing Surveys* 23.1, pp. 5–48. URL: https://doi.org/10.1145/103162.103163.

Golub, Gene H. and Charles F. Van Loan (2013). *Matrix Computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Baltimore, MD: Johns Hopkins University Press, pp. xiv+756.

Good, Irving John (1958). "The Interaction Algorithm and Practical Fourier Analysis". In: *Journal of the Royal Statistical Society*. B 20, pp. 361–372.

— (1960). "The Interaction Algorithm and Practical Fourier Analysis: An Addendum". In: *Journal of the Royal Statistical Society*. B 22, pp. 372–375.

Han, Song, Jeff Pool, John Tran, and William J. Dally (2015). "Learning Both Weights and Connections for Efficient Neural Networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: MIT Press, 1135–1143.

Haselman, M., M. Beauchamp, A. Wood, S. Hauck, K. Underwood, and K.S. Hemmert (2005). "A comparison of floating point and logarithmic number systems for FPGAs". In: *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*, pp. 181–190.

He, Kaiming and Jian Sun (2015). "Convolutional neural networks at constrained time cost". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 5353–5360.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034.

— (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Higham, Nicholas J. (2002). *Accuracy and Stability of Numerical Algorithms*. 2nd. Philadelphia, PA: SIAM Publications.

Hubel, David H. and Torsten N. Wiesel (1959). "Receptive Fields of Single Neurons in the Cat's Striate Cortex". In: *Journal of Physiology* 148, pp. 574–591.

Huffman, David (1952). "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9, pp. 1098–1101.

Ju, Caleb and Edgar Solomonik (2020). "Derivation and Analysis of Fast Bilinear Algorithms for Convolution". In: *SIAM Rev.* 62, pp. 743–777.

Kahan, William (1996). "The Improbability of Probabilistic Error Analyses for Numerical Computations". URL: http://people.eecs.berkeley.edu/~wkahan/improber.pdf.

Karatsuba, Anatoly (1995). "The Complexity of Computations". In: *Proceedings of the Steklov Institute of Mathematics* 211, pp. 169–183.

Karatsuba, Anatoly and Yuri Ofman (1962). "Multiplication of many-digital numbers by automatic computers". In: *Proceedings of the USSR Academy of Sciences* 145, pp. 293–294.

— (1963). "Multiplication of many-digital numbers by automatic computers". In: *Physics-Doklady* 7, pp. 595–596.

Kloss Carey, Intel VP Hardware and AI Products Group (2019). *Deep Learning By Design; Building silicon for AI*.

Knuth, Donald E. (1998). *The Art of Computer Programming*. USA: Addison-Wesley.

Krizhevsky, Alex (2009). *Learning multiple layers of features from tiny images*. Tech. rep.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Lang, Kevin J. (1988). "Connectionist speech recognition". PhD thesis. Pittsburgh, PA.

Lang, Kevin J., Alex Waibel, and Geoffrey E. Hinton (1990). "A time-delay neural network architecture for isolated word recognition". In: *Neural Networks* 3.1, pp. 23–43.

Lavin, Andrew. URL: https://github.com/andravin/wincnn.

Lavin, Andrew and Scott Gray (2016). "Fast algorithms for convolutional neural networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recog-*

nition (CVPR). Las Vegas, Nevada: IEEE, pp. 4013–4021. URL: https://doi.org/10.1109/CVPR.2016.435.

Le, Quoc V., Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeffrey Dean, and Andrew Y. Ng (2012). "Building high-level features using large scale unsupervised learning". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. URL: http://icml.cc/2012/papers/73.pdf.

LeCun, Yann, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel (1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1, pp. 541–551.

LeCun, Yann, Corinna Cortes, and C.J.Burges (2010). "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2.

Liu, Junhong, Dongxu Yang, and Junjie Lai (2021). "Optimizing Winograd-Based Convolution with Tensor Cores". In: *50th International Conference on Parallel Processing*. New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3472456.3472473.

Liu, Xingyu, Jeff Pool, Song Han, and William J. Dally (2018). "Efficient Sparse-Winograd Convolutional Neural Networks". In: *ICLR (Poster)*. OpenReview.net.

Liu, Zhi-Gang and Matthew Mattina (2020). "Efficient Residue Number System Based Winograd Convolution". In: *ECCV*.

Maji, Partha P. and R. Mullins (2018). "On the Reduction of Computational Complexity of Deep Convolutional Neural Networks †". In: *Entropy* 20.

Maji, Partha P., Andrew Mundy, Ganesh S. Dasika, Jesse G. Beu, Matthew Mattina, and R. Mullins (2019). "Efficient Winograd or Cook-Toom Convolution Kernel Implementation on Widely Used Mobile CPUs". In: *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pp. 1–5.

Mcculloch, Warren and Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5, pp. 127–147.

McGinty, Brendan (2018). *Deep learing at scale*. URL: https://verneglobal.com/news/blog/deep-learning-at-scale.

Meng, Lingchuan and John Brothers (2019). "Efficient Winograd Convolution via Integer Arithmetic". In: *CoRR* abs/1901.01965. arXiv: 1901.01965.

Nussbaumer, Henri J. (1981). *Fast Fourier Transform and Convolution Algorithms*. Springer-Verlag.

P754, IEEE Task (1985). *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. pub-IEEE-STD, p. 20.

Pan, Victor Y. (2016). "How bad are Vandermonde matrices?" In: *SIAM J. Matrix Analysis Applications* 37.2, pp. 676–694. URL: https://doi.org/10.1137/15M1030170.

Parhami, Behrooz (2020). "Computing with logarithmic number system arithmetic: Implementation methods and performance benefits". In: *Comput. Electr. Eng.* 87, p. 106800. URL: https://doi.org/10.1016/j.compeleceng.2020.106800.

Parhi, Keshab K. (2007). *VLSI Digital Signal Processing Systems: Design and Implementation*. New York: John Wiley and Sons.

Reuther, Albert, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner (2020). "Survey of Machine Learning Accelerators". In: *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. URL: http://dx.doi.org/10.1109/HPEC43674.2020.9286149.

Rudin, Walter (May 1986). *Real and Complex Analysis*. McGraw-Hill Science/Engineering/Math. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0070542341.

Rump, Siegfried M., Takeshi Ogita, and Shin'ichi Oishi (2008a). "Accurate floating-point summation Part I: faithful rounding". In: *SIAM Journal on Scientific Computing* 31.1, pp. 189–224. URL: https://doi.org/10.1137/050645671.

— (2008b). "Accurate floating-point summation part II: sign, K-fold faithful and rounding to nearest". In: *SIAM Journal on Scientific Computing* 31.2, pp. 1269–1302. URL: https://doi.org/10.1137/07068816X.

Selesnick, Ivan W. and C. Sidney Burrus (1994). "Extending Winograd's Small Convolution Algorithm to Longer Lengths". In: *1994 IEEE International Symposium on Circuits and Systems, ISCAS 1994, London, England, UK, May 30 - June 2, 1994*, pp. 449–452.

Sergio Guadarrama, Nathan Silberman (2016). *TensorFlow-Slim: A lightweight library for defining, training and evaluating complex models in TensorFlow*. https://github.com/google-research/tf-slim. URL: https://github.com/google-research/tf-slim.

Steele, Michael J. (2004). *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. Cambridge University Press.

Swartzlander, Earl E. and Aristides G. Alexopoulos (1975). "The Sign/Logarithm Number System". In: *IEEE Transactions on Computers* C-24, pp. 1238–1242.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions". In: *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9.

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf (2014). "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1701–1708.

Thomas, Llewellyn H. (1963). "Using a Computer to Solve Problems in Physics". In: *Applications of Digital Computers*.

Tolimieri, Richard, Myoung An, and Chao Lu (1997). *Algorithms For Discrete Fourier Transform and Convolution*. 2nd. New York, NY, USA: Springer-Verlag.

Toom, Andrei L. (1963). "The complexity of a scheme of functional elements realizing multiplication of integers". In: *Soviet Mathematics – Doklady* 3, pp. 714–716.

Trefethen, Lloyd N. and David Bau (1997). *Numerical Linear Algebra*. Philadelphia: SIAM.

Vijh, Surbhi, Adesh Kumar Pandey, Garima Vijh, and Sumit Kumar (2021). "Stock Forecasting for Time Series Data using Convolutional Neural Network". In: *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pp. 866–870.

Vincent, Kevin, Kevin Stephano, Michael Frumkin, Boris Ginsburg, and Julien Demouth (2017). "On improving the numerical stability of Winograd Convolutions". In: *Proceedings of the 5th International Conference on Learning Representations*. Toulon, France, p. 4.

Waibel, A., T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang (1989). "Phoneme recognition using time-delay neural networks". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3, pp. 328–339.

Wang, Wei and Jianxun Gang (2018). "Application of Convolutional Neural Network in Natural Language Processing". In: *2018 International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pp. 64–70.

Wilkinson, James H. (1994). *Rounding Errors in Algebraic Processes*. New York, NY, USA: Dover Publications.

Winograd, Shmuel (1976). "On Computing the Discrete Fourier Transform". In: *Proceedings of the National Academy of Sciences of the United States of America* 73, pp. 1005–1006.

— (1978). "On Computing the Discrete Fourier Transform". In: *Mathematics of Computation* 32, pp. 175–199.

— (1980a). *Arithmetic Complexity Computations*. Bristol, England: SIAM Publications.

— (1980b). "Signal processing and complexity of computation". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP*. Denver, Colorado: IEEE, pp. 94–101. URL: https://doi.org/10.1109/ICASSP.1980.1171044.

Young Cliff, Google AI (2018). *Codesign in Google TPUs: Inference, Training, Performance, and Scalability*.

Zlateski, Aleksandar, Zhen Jia, Kai Li, and Frédo Durand (2018). "FFT Convolutions are Faster than Winograd on Modern CPUs, Here is Why". In: *CoRR*

abs/1809.07851. arXiv: 1809.07851. URL: http://arxiv.org/abs/1809.07851.