A Comparison of Different Algorithms for Sparse Einsum

BACHELOR THESIS

to be Awarded the Academic Degree

Bachelor of Science (B.Sc.)

in Informatics

FRIEDRICH-SCHILLER-UNIVERSITY
JENA
Faculty for Mathematics and Informatics

Submitted by Leon Manthey
born on 31.10.1999 in Berlin
Supervisor: Mark Blacher

Jena, 15.05.2024

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce eleifend orci et venenatis cursus. Nullam eget ornare lacus. Donec non dolor non tellus eleifend vehicula. Sed et lorem lectus. Vestibulum sagittis sed nisi ac interdum. Duis nec accumsan velit, hendrerit malesuada magna. Aliquam erat volutpat. Cras eu ante nec est malesuada volutpat. Proin quis posuere quam. Etiam aliquam eros quis dui sagittis, a fermentum lectus rutrum. Nunc tempor mauris vel tellus facilisis rhoncus. Aliquam ut leo eget metus volutpat vestibulum. Nam non consequat ante. In rutrum felis in enim fringilla lacinia. Phasellus ut imperdiet risus. Curabitur tincidunt libero sed urna dignissim, eget rutrum felis scelerisque. Nunc ut convallis neque, non tincidunt nulla. Curabitur quis condimentum leo. Phasellus laoreet ligula vel mi commodo, id accumsan diam tristique. Maecenas euismod lorem in tempor iaculis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut eget purus sem. Suspendisse venenatis aliquet dignissim. Integer turpis lorem, tempus non turpis et, gravida aliquet erat. Sed vel neque non ex ultrices vestibulum. Aliquam purus quam, rhoncus non ante at, convallis sagittis erat. Sed justo elit, vulputate vel accumsan non, porta eget turpis. Proin eget ultrices sem. Nunc eu velit.

Contents

1 Introduction

Einstein summation notation is a powerful and compact notation for representing tensor expressions. It was introduced by Albert Einstein in the early 20th century as a means to simplify tensor expressions in the theory of relativity [2]. The notation is both elegant and efficient, making it a valuable tool in various fields such as theoretical physics, computational mathematics, and data science.

The fundamental operation in Einstein summation notation is the Einstein summation, often referred to simply as "Einsum". This operation allows for the calculation of various tensor operations, including element-wise multiplication, dot products, outer products, and matrix multiplications expressed in Einstein notation. The computational efficiency and expressiveness of Einsum have led to its adoption in numerous applications, ranging from machine learning to scientific computing.

In many practical applications, especially in machine learning and scientific computing, the data involved is often sparse. In sparse tensors most values are zero. Handling sparse tensors efficiently requires specialized algorithms and data structures to avoid unnecessary computations and to save memory. Traditional libraries like NumPy [3] and other major artificial intelligence frameworks [4, 6] typically support Einstein summation for dense tensors, but not for sparse tensors. The only known library that aims to support Einsum operations on sparse tensors is Sparse [7]. However, like NumPy, Sparse only allows for a limited number of symobls as to be used as indices, which is why we use the package opt_einsum [8]. opt_einsum is a package for optimizing the contraction order of Einsum expressions. More importantly for us though, opt_einsum can handle UTF-8 symbols and use Sparse and other libraries like Torch as a backend. Real Einstein summation problems often inculde expressions with hundreds or even thousands of higher order tensors. In order to express the aforementioned operations we require a large set of unique symbols. Thus, our approaches, just like opt_einsum, are capable of handling all symbols in the UTF-8 encoding.

This thesis explores the implementation and performance of Einstein summation across different computing paradigms, with a particular focus on sparse tensors. Specifically, it focuses on explaining our following implementations and comparing them to multiple libraries:

- SQL-based Implementation: This implementation is based on the algorithm presented in "Efficient and Portable Einstein Summation in SQL" by Blacher et al [1]. It constructs SQL queries dynamically using Python. While SQL is traditionally used for database operations, this approach demonstrates the versatility of SQL in performing tensor operations.

- C++ Implementation: The second implementation is written in C++, with multiple versions ranging from naive to optimized approaches. The different versions aim to explore the performance trade-offs between simplicity and optimization, offering insights into how different coding strategies affect computational efficiency.

By comparing these implementations, we aim to provide a comprehensive analysis of the performance and scalability of sparse Einstein summation in various computing environments. The SQL-based implementation serves as a baseline, showcasing the potential of database query languages for tensor operations. Furthermore, the C++ implementations demonstrate the impact of low-level optimizations on computational performance. Comparing these against the sparse library Sparse and highly performance-tuned dense libraries like PyTorch provides insights into different use cases and helps identify the optimal tool for various tasks. The code for the tools can be found on Github https://github.com/Lethey2552/Sparse-Einsum.
Through this comparative study, we seek to identify the strengths and weaknesses of each approach, providing guidelines for selecting the appropriate method based on specific use cases and computational requirements. This work contributes to the broader understanding of tensor operations and their efficient implementation, offering practical insights for researchers and practitioners in fields that rely heavily on tensor computations.

2 Background

The following chapter serves to introduce the necessary background for tensors, Einstein notation and Einstein summation. We will also explain tensor hypernetworks and various operations that can be expressed using Einstein notation. Due to a considerable overlap of topics we will use the background section of "Optimizing Tensor Contraction Paths: A Greedy Algorithm Approach With Improved Cost Functions" [5] as a guideline and extended it for our uses.

2.1 Tensors

Tensors are algebraic objects and a fundamental concept in mathematics, physics and computer science. They extend the idea of scalars, vectors and matrices to higher dimensions. In essence, a tensor is a multi-dimensional array with an abitrary number of dimensions.
Each dimension of a tensor is represented by an index with its own range. The number of indices is commonly referred to as the tensor's "rank" or "order." The size of a tensor is determined by the product of the maximum values of each index's range.
For example, consider a tensor T with indices i, j, k and corresponding ranges $i \in \{1, 2\},\ j \in \{1, 2, 3, 4, 5, 6\}$ and $k \in \{1, 2, 3, 4\}$. The size of tensor T is calculated as follows: $2 \cdot 6 \cdot 4 = 48$. This means tensor T has a total of 48 elements.

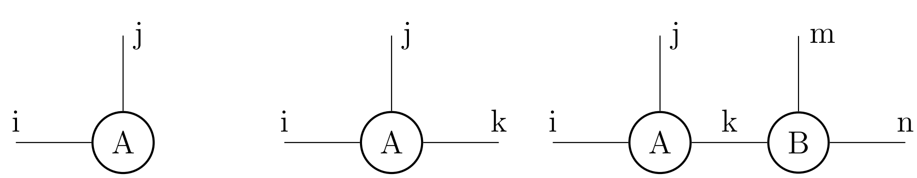

Figure 2.1: A matrix, a tensor and a tensor network visualized as a graph. Each index is represented by an edge. Shared indices in a tensor network are edges between nodes.

2.1.1 Sparse Tensors

In Sparse tensors most values are zero. They can greatly profit from specialized formats. For our tensor $T \in \mathbb{R}^{I \times J \times K}$ in dense format we need to save $I \cdot J \cdot K$ values no matter whether they are zero or not. Now consider that, if the vast majority of Ts values are zero, we could only save the coordinate of the value for each dimensions and the value itself. This is what we call the coordinate (COO) format. A sparse tensor could be reduced to the COO format as follows:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 5 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 4 \\ 2 & 0 & 5 \\ 2 & 3 & 10 \end{bmatrix}$$

Each row of the COO representation encodes a single value of the matrix with each column giving the position of the value in each dimension and the last column giving the actual value.

2.2 Einstein Notation and Summation

In 1916, Albert Einstein introduced the so called Einstein notation, also known as Einstein summation convention or Einstein summation notation, for the sake of representing tensor expressions in a concise manner. As an example, the contraction of tensors $A \in \mathbb{R}^{I \times J \times K}$ and $B \in \mathbb{R}^{K \times M \times N}$ in figure 2.1,

$$C_{ijmn} = \sum_k A_{ijk} \cdot B_{kmn}$$

can be simplified by making the assumption that pairs of repeated indices in the expression are to be summed over. Consequently, the contraction can be rewritten as:

$$C_{ijmn} = A_{ijk} \cdot B_{kmn}$$

To expand upon the expressive power of the original Einstein notation, modern Einstein notation was introduced. This notation is used by most linear algebra and machine learning libraries that provide Einstein summation notation. Modern Einstein notation explicitly states the indices for the output tensor, enabling further operations like transpositions and traces.
In modern Einstein notation, the expression from our previous example would be written as:

$$A_{ijk} B_{kmn} \rightarrow C_{ijmn}$$

When using common Einstein summation APIs, tensor operations are encoded by using the indices of the tensors in a format string and the data itself.

The format string for the above operation would come down to:

$$ijk, kmn \rightarrow ijmn$$

In Modern Einstein notation, indices that are not mentioned in the output are to be summed over. For the sake of simplicity, we will from now on refer to Einstein summation as Einsum, and we will use the original, the modern notation or just the format string, depending on the context.

2.3 Tensor Hypernetworks

2.4 Operations with Einsum

Einsum is a powerful tool for performing various tensor operations concisely and efficiently. Here are some common operations that can be performed using Einsum:

Operation	**Formula**	**Einsum Notation**
Matrix Multiplication	$C_{ij} = \sum_k A_{ik}B_{kj}$	ik,kj $\rightarrow$ ij
Dot Product	$c = \sum_i a_i b_i$	i,i $\rightarrow$
Outer Product	$C_{ij} = a_i b_j$	i,j $\rightarrow$ ij
Tensor Contraction	$C_{kl} = \sum_{ij} A_{kij}B_{ijl}$	kij,ijl $\rightarrow$ kl
Transpose	$B_{ji} = A_{ij}$	ij $\rightarrow$ ji
Sum Over Axes	$b_j = \sum_i A_{ij}$	ij $\rightarrow$ j

Table 2.1: Common Operations with Einsum

These examples illustrate the versatility and power of the Einsum function in performing a wide range of tensor operations with concise and readable notation.

Bibliography

[1] Mark Blacher et al. “Efficient and Portable Einstein Summation in SQL”. In: *Proc. ACM Manag. Data* 1.2 (2023).

[2] Albert Einstein. “Die Grundlage der allgemeinen Relativitätstheorie”. In: *Annalen der Physik*. Vierte Folge Band 49 (1916). pp. 781-782, pp. 769–822.

[3] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (2020), pp. 357–362.

[4] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: `https://www.tensorflow.org/`.

[5] Sheela Orgler and Mark Blacher. *Optimizing Tensor Contraction Paths: A Greedy Algorithm Approach With Improved Cost Functions*. 2024. arXiv: `2405.09644 [quant-ph]`. URL: `https://arxiv.org/abs/2405.09644`.

[6] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *CoRR* abs/1912.01703 (2019).

[7] Hameer Rocklin Matthew Abbasi. *Sparse 0.15.4*. `https://github.com/pydata/sparse`. 2024.

[8] Daniel G. a. Smith and Johnnie Gray. “opt_einsum - A Python package for optimizing contraction order for einsum-like expressions”. In: *Journal of Open Source Software* 3.26 (2018), p. 753. URL: `https://doi.org/10.21105/joss.00753`.