FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Exploring Einsum as a Universal Inference Language

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

im Studiengang Informatik

Friedrich-Schiller-Universität Jena

Fakultät für Mathematik und Informatik

von Maurice Wenig **geboren am** 11.08.1999 **in** Jena

Betreuer: Prof. Dr. Joachim Giesen, Dr. Julien Klaus

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Contents

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2 Einsum

Given two third-order tensors $A \in \mathbb{R}^{3\times4\times5}$ and $B \in \mathbb{R}^{3\times3\times5}$, and a vector $v \in \mathbb{R}^4$. Consider the following computation resulting in a matrix $C \in \mathbb{R}^{3\times3}$:

$$\forall i \in [3] : \forall j \in [4] : C_{ij} = \sum_{k=1}^{5} A_{ijk} B_{iik} v_j$$

The original Einstein-notation for summation removes redundant formalism ("boilerplate") from this expression:

$$C_{ij} = A_{ijk} B_{iik} v_j$$

where it is assumed that C is defined for all possible i, j. We sum over all indices that are not used to index the output. In this example, we therefore have to sum over all possible values of k, because it is not used to index C_{ij}. Note how it is clear what the shape of C is, because i and j were used to index the tensors A, B, and v, for which we defined the dimensions on every axis.

This notation is essentially the inspiration for Einsum, which might be apparent given the name Einsum. Einsum is just an adaptation of this style, which makes it easier to use in programming. With it, we can write the above expression like this:

$$C := (ijk, iik, j \to ij, A, B, v)$$

Through the following definition, we hope to clear up why this Einsum expression results in the computation above, and what computation is the result of a general Einsum expression.

2.1 Syntax and Semantics

Definition 1. Einsum expressions specify how several input tensors are combined into a single output tensor. Let $T^{(1)}, \ldots, T^{(n)}$ be our input tensors, where $T^{(i)}$ is an n_i-th order tensor for $i \in [n]$. The core of the Einsum expression are index strings. For this, we first need a collection of symbols S. The respective index string for a tensor $T^{(i)}$ is then just a tuple $\boldsymbol{s_i} \in S^{n_i}$, composed of symbols $s_{ij} \in S$ for $j \in [n_i]$. The index string that is right of the arrow ($\to$) belongs to the output tensor T and is referred to as output string $\boldsymbol{s_t}$.

In our example this could be $S = \{i, j, k\}$. The tensor $T^{(1)} = A$ has the index string $\boldsymbol{s_1} = ijk$, $T^{(2)} = B$ has $\boldsymbol{s_2} = iik$, $T^{(3)} = v$ has $\boldsymbol{s_3} = j$, and the output string is $\boldsymbol{s_t} = ij$. The individual symbols are $s_{11} = i$, $s_{12} = j$, $s_{13} = k$, $s_{12} = i$, $s_{22} = i$, $s_{23} = k$, $s_{31} = j$, $s_{t1} = i$, $s_{t2} = j$.

The next step in the definition is to speak about axis sizes. If we want to iterate over shared indices, it is necessary that the axes, that these indices are used for, share the same size. In our example, A_{ijk} and v_j share the symbol $s_{12} = s_{31} = j$. This means that the second axis of A and the first axis of v have to have the same size, which happens to be four. Let us express this formally.

Let $d_{ij} \in \mathbb{N}$ denote the size of the j-th axis of $T^{(i)}$ for $i \in [n], j \in [n_i]$. Then it must hold that $s_{ij} = s_{i'j'} \implies d_{ij} = d_{i'j'}$ for all $i, i' \in [n], j \in [n_i], j' \in [n_{i'}]$.

Therefore we can also denote the size of all axes, that a symbol $s \in S$ corresponds to, as $d_s := d_{ij}$ for all $i \in [n], j \in [n_i]$ with $s = s_{ij}$. Note that not all same size axes have to be assigned the same symbol. For instance a square matrix could have index strings $\boldsymbol{s} = (i, i)$ or $\boldsymbol{s} = (i, j)$.

The next step of the definition is figuring out which symbols are used for summation and which symbols are used for saving the result of the computation. In order to do this, it is useful to know which symbols are in an index string, because symbols can occur more than once in just one index string (as seen in B_{iik} in our example). Therefore, let $\sigma(\boldsymbol{s})$ denote the set with all symbols used in an index string $\boldsymbol{s}$. That is, in out example $\sigma(\boldsymbol{s_2}) = \sigma(iik) = \{i, k\}$.

All symbols to the right of the arrow ($\rightarrow$) are used as an index for the result of the computation. These symbols are called *free* symbols $F = \sigma(\boldsymbol{s_t})$. All other symbols used in the expression are called *bound* symbols $B = \bigcup_{i \in [n]} \sigma(\boldsymbol{s_i}) \setminus \sigma(\boldsymbol{s_t})$. The reasoning behind this name is, that these symbols are bound by the summation symbol in the original computation. In Einsum, we sum over all axes that belong to bound symbols. It follows that the multi-index space that we iterate over is $\mathcal{F} = \prod_{s \in F}[d_s]$ and the multi-index space we sum over is $\mathcal{B} = \prod_{s \in B}[d_s]$. In our example, the free symbols are $F = \{ij\}$ and the bound symbols are $B = \{k\}$. The multi-index space we iterate over is $d_i \times d_j = [3] \times [4]$. The multi-index space we sum over is $d_k = [5]$.

From the definition of $\mathcal{F}$, it follows that d_s has to be defined for all symbols $s \in F$. This means we have to add the constraint $\sigma(\boldsymbol{s_t}) \subseteq \bigcup_{i \in [n]} \sigma(\boldsymbol{s_i})$.

However, we do not use every symbol in the multi-index spaces to index every input tensor. Instead, we use the index strings $\boldsymbol{s}$ to index the tensor. To formally express this, we need a projection from a multi-index $(\boldsymbol{f}, \boldsymbol{b}) \in \mathcal{F} \times \mathcal{B}$ [1] to another multi-index, which includes only the indices, that are represented by the symbols used in $\boldsymbol{s}$, in the same order as present in $\boldsymbol{s}$. We denote this as $(\boldsymbol{f}, \boldsymbol{b}) : \boldsymbol{s}$. Notice how this still allows duplication of indices given in $(\boldsymbol{f}, \boldsymbol{b})$. This is needed, as can be seen in our example for B_{iik}, where a multi-index, e.g. $(i = 1, j = 4, k = 2) \in \mathcal{F} \times \mathcal{B}$, is projected onto a

[1] Here, we use $(\boldsymbol{f}, \boldsymbol{b})$ as the notation for concatenating the tuples $\boldsymbol{f}$ and $\boldsymbol{b}$. This means, $(\boldsymbol{f}, \boldsymbol{b})$ is not a tuple of multi-indices, but another multi-index.

different multi-index, by the index string iik. With this index string, the index that is represented by the symbol i is projected onto the first and second position, and the index that is represented by the symbol k is projected onto the third position. Therefore, the resulting multi-index is $(i = 1, j = 4, k = 2) : iik = (1, 1, 2)$.

In our example, we used the standard sum and multiplication as operators for computing our result. But with Einsum, we allow the more general use of any semiring $R = (M, \oplus, \odot)$. With this, we can finally define a general Einsum expression

$$T := (\boldsymbol{s_1}, \ldots, \boldsymbol{s_n} \to \boldsymbol{s_t}, T^{(1)}, \ldots, T^{(n)})_R$$

in terms of semiring operations. Namely, T is the $|\boldsymbol{s_t}|$-th order tensor

$$\forall \boldsymbol{f} \in \mathcal{F} : T_{\boldsymbol{f}:\boldsymbol{s_t}} = \bigoplus_{\boldsymbol{b} \in \mathcal{B}} \bigodot_{i=1}^{n} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}}.$$

Because we also project the indices $\boldsymbol{f}$ with the output string $\boldsymbol{s_t}$, we allow to iterate over duplicate indices, e.g. $\mathrm{diag}(v) = (j \to jj, v)$. This leaves some entries of the result undefined. We define these entries to be the additive neutral element in the given semiring R. This may sound arbitrary at first, but will be useful later.

In case the semiring can be derived from the context, or if it is irrelevant, it can be left out from the expression.

The careful reader might have noticed two potential problems that could arise in the above definition. The first potential problem could arise when one of the input tensors is a scalar, which is a 0-th order tensor. This would mean that the index string $\boldsymbol{s}$ for that input tensor has to be the empty string ϵ. Now when the multi-index $(\boldsymbol{f}, \boldsymbol{b})$ is projected by this empty index string, then the resulting multi-index can only be the empty multi-index $\lambda := ()$. One might expect that this leads to a problem, because we can not access any entries of a tensor with an empty multi-index. But for scalars, it makes sense to define the empty multi-index in such a way, that it accesses precisely the only entry that is stored in the scalar, i.e. $T_\lambda := T$ for a scalar T. This way, we can easily support scalars with empty index strings in Einsum.

The second potential problem could arise when either the free symbols F or the bound symbols B are empty, because the universal quantor over an empty multi-index space $\mathcal{F}$ is always trivially true, and the sum over an empty multi-index space $\mathcal{B}$ is always trivially zero. But in fact, this leads to no problem, because the induced multi-index spaces of empty F or B are not empty themselves. They contain one element, namely the set including only the empty multi-index $\{\lambda\}$. In the following, we will explain why this is the case, and how this solves any problems with empty F or B.

Notice the definition of the product we use for to sets M, N:

$$M \times N = \{(m, n) \mid m \in M, n \in N\}.$$

This looks like an ordinary cartesian product, but the hidden difference lies in the meaning of (m, n). Namely, if m and n are multi-indices $\boldsymbol{m} = (m_1, \ldots, m_{k_1})$ and $\boldsymbol{n} = (n_1, \ldots, n_{k_2})$ for $k_1, k_2 \in \mathbb{N}$, then we defined $(\boldsymbol{m}, \boldsymbol{n})$ to be the concatenation of the multi-indices:

$$(\boldsymbol{m}, \boldsymbol{n}) = (m_1, \ldots, m_{k_1}, n_1, \ldots, n_{k_2}),$$

which is one tuple with the entries of $\boldsymbol{m}$ and $\boldsymbol{n}$, instead of the tuple of tuples

$$((m_1, \ldots, m_{k_1}), (n_1, \ldots, n_{k_2})).$$

Therefore we can name a neutral element for concatenation, which is the empty multi-index λ with $(\boldsymbol{i}, \lambda) = \boldsymbol{i}$ for any multi-index $\boldsymbol{i}$. From this, we can derive a neutral element for our product of multi-index spaces, which is the set including only the empty multi-index $\{\lambda\}$ with $\mathcal{I} \times \{\lambda\} = \mathcal{I}$ for any multi-index space $\mathcal{I}$.

Now, because it makes sense to define an operation over an empty set of operands as the neutral element of said operation, we can safely define

$$\prod_{i \in \emptyset} [d_i] := \{\lambda\}\,.$$

Therefore, if $F = \emptyset$, then

$$T := (\boldsymbol{s_1}, \ldots, \boldsymbol{s_n} \to \boldsymbol{s_t}, T^{(1)}, \ldots, T^{(n)})_R$$

results in the computation of a $|\boldsymbol{s_t}|$-th order tensor T with

$$\begin{aligned} &\forall \boldsymbol{f} \in \mathcal{F} : T_{\boldsymbol{f}:\boldsymbol{s_t}} = \sum_{\boldsymbol{b} \in \{\lambda\}} \bigodot_{i=1}^{n} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}} \\ \iff &\forall \boldsymbol{f} \in \mathcal{F} : T_{\boldsymbol{f}:\boldsymbol{s_t}} = \bigodot_{i=1}^{n} T^{(i)}_{\boldsymbol{f}:\boldsymbol{s_i}}. \end{aligned}$$

And if $B = \emptyset$, then

$$T := (\boldsymbol{s_1}, \ldots, \boldsymbol{s_n} \to, T^{(1)}, \ldots, T^{(n)})_R$$

results in the computation of a scalar T with

$$\begin{aligned} &\forall \boldsymbol{f} \in \{\lambda\} : T_{\boldsymbol{f}:\epsilon} = \bigoplus_{\boldsymbol{b} \in \mathcal{B}} \bigodot_{i=1}^{n} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}} \\ \iff &T = \bigoplus_{\boldsymbol{b} \in \mathcal{B}} \bigodot_{i=1}^{n} T^{(i)}_{\boldsymbol{b}:\boldsymbol{s_i}}. \end{aligned}$$

2.2 Simple Examples From Linear Algebra

All following examples use the standard semiring $R = (\mathbb{R}, +, \cdot)$.

- matrix-vector multiplication: Let $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$. Then

$$A \cdot v = (ij, j \to i, A, v)$$

- matrix-matrix multiplication: Let $A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}$. Then

$$A \cdot B = (ik, kj \to ij, A, B)$$

- trace: Let $A \in \mathbb{R}^{n \times n}$. Then

$$\text{trace}(A) = (ii \to, A)$$

- squared Frobenius norm: Let $A \in \mathbb{R}^{n \times n}$. Then

$$|A|_2^2 = (ij, ij \to, A, A)$$

- diagonal matrix: Let $v \in \mathbb{R}^n$. Then

$$\text{diag}(v) = (i \to ii, v)$$

2.3 Computational Benefits

Query Language, faster inference, stuff like that. How can it be faster? - we can optimize contraction path - high parallelism allows for a lot of optimization with the use of vectorization - intermediate steps in optimal contraction path can be optimized with pipelining - we can optimize hardware specifically for computing Einsum expressions, which is pretty much just tensor operations.

By mapping a lot of problems to Einsum, we can spare time in trying to find efficient solutions for new problems, and focus on optimizing the computation of Einsum expressions. This reduces the effort of finding an efficient solution for a problem to finding a mapping of the problem to einsum.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis

non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3 Nested Expressions

In practice, concatenations of operations come naturally, e.g. computing the squared norm of a matrix-vector product $|A \cdot v|_2^2$ for $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$. This would lead to a nested Einsum expression $|A \cdot v|_2^2 = (i, i \to, (ij, j \to i, A, v), (ij, j \to i, A, v))$. This expression dictates the order of evaluating the expression. In the example of the norm, the expression $(ij, j \to i, A, v)$ has to be evaluated before squaring and summing over the results of this computation.

This is limiting, because the order of evaluation might not yield optimal runtime. This can be seen with a simple matrix-matrix-vector multiplication, which can be written as follows:

$$(A \cdot B) \cdot v = (ij, j \to i, (ik, kj \to ij, A, B), v)$$

which is clearly worse than the optimal contraction order

$$A \cdot (B \cdot v) = (ij, j \to i, A, (ij, j \to i, B, v))$$

for $A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}, v \in \mathbb{R}^n$. Another limitation of nested Einsum strings is that we can not fully benefit from the computational advantages of a single Einsum string, that were stated in Section 2.3.

But fortunately, all nested Einsum expressions can be compressed into a single Einsum expression, if they are computed over the same semiring. For instance,

$$\begin{aligned} |A \cdot v|_2^2 &= (i, i \to, (ij, j \to i, A, v), (ij, j \to i, A, v)) \\ &= (ij, j, ij, j \to, A, v, A, v) \end{aligned}$$

and

$$\begin{aligned} (A \cdot B) \cdot v &= (ij, j \to i, (ik, kj \to ij, A, B), v) \\ &= (ik, kj, j \to i, A, B, v). \end{aligned}$$

This leaves the path of contraction up to the implementation, and lets us benefit from all the computational advantages mentioned in Chapter 2. In the following theorems, we assume that the computations are all over the same semiring $R = (M, \oplus, \odot)$.

3.1 Simple Nested Expressions

In the following, we will explore how to compress such expressions as

$$\underbrace{(ij, j \to i, \overbrace{(ik, kj \to ij, A, B)}^{\text{inner expression}}, v)}_{\text{outer expression}}$$

for $A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}, v \in \mathbb{R}^n$.

Theorem 1: For $i \in [m+n]$, let $T^{(i)}$ be an n_i-th order tensor with index strings $\boldsymbol{s_i} \in S^{n_i}$. Let $\boldsymbol{s_u}, \boldsymbol{s_v}$ be index strings. Let

$$U := (\boldsymbol{s_{m+1}}, \dots, \boldsymbol{s_{m+n}} \to \boldsymbol{s_u}, T^{(m+1)}, \dots, T^{(m+n)})$$

and

$$V := (\boldsymbol{s_1}, \dots, \boldsymbol{s_m}, \boldsymbol{s_u} \to \boldsymbol{s_v}, T^{(1)}, \dots, T^{(m)}, U)$$

where the bound symbols of the second Einsum expression share no symbols with the first Einsum expression. Then

$$V = (\boldsymbol{s_1}, \dots, \boldsymbol{s_{m+n}} \to \boldsymbol{s_v}, T^{(1)}, \dots, T^{(m+n)})$$

Proof. Let F, F', B, B' be the free and bound symbols of the outer and inner Einsum expression respectively. W.l.o.g. they are all non-empty. From them we can derive the multi-index spaces $\mathcal{F}, \mathcal{F}', \mathcal{B}, \mathcal{B}'$ as in the definition. Then

$$\begin{aligned}
V & = (\boldsymbol{s_1}, \dots, \boldsymbol{s_m}, \boldsymbol{s_u} \to \boldsymbol{s_v}, T^{(1)}, \dots, T^{(m)}, U) \\
\iff \forall \boldsymbol{f} \in \mathcal{F} : V_{\boldsymbol{f}:\boldsymbol{s_v}} & = \bigoplus_{\boldsymbol{b} \in \mathcal{B}} \bigodot_{i=1}^{m} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_k}} \odot U_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_u}} \\
& = \bigoplus_{\boldsymbol{b} \in \mathcal{B}} \bigodot_{i=1}^{m} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}} \odot \left[\bigoplus_{\boldsymbol{b'} \in \mathcal{B}'} \bigodot_{i'=m+1}^{m+n} T^{(i')}_{(\boldsymbol{f},\boldsymbol{b},\boldsymbol{b'}):\boldsymbol{s_{i'}}} \right] \\
& = \bigoplus_{\boldsymbol{b} \in \mathcal{B}} \bigoplus_{\boldsymbol{b'} \in \mathcal{B}'} \bigodot_{i=1}^{m} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}} \odot \bigodot_{i=m+1}^{m+n} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b},\boldsymbol{b'}):\boldsymbol{s_i}} \\
& = \bigoplus_{\boldsymbol{b} \in \mathcal{B} \times \mathcal{B}'} \bigodot_{i=1}^{m+n} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}} \\
\iff V & = (\boldsymbol{s_1}, \dots, \boldsymbol{s_{m+n}} \to \boldsymbol{s_v}, T^{(1)}, \dots, T^{(m+n)})
\end{aligned}$$

where the third equality follows from the definition of U:

$$\forall \boldsymbol{f'} \in \mathcal{F}' : U_{\boldsymbol{f'}:\boldsymbol{s_u}} = \bigoplus_{\boldsymbol{b'} \in \mathcal{B}'} \bigodot_{i'=m+1}^{m+n} T^{(i')}_{(\boldsymbol{f'},\boldsymbol{b'}):\boldsymbol{s_{i'}}}$$

and from the fact, that the symbols in $\boldsymbol{s_u}$ are used in the outer expression as an input string, and in the inner expression as the output string, and therefore $F' \subseteq B \cup F$. Additionally, because of the stated requirement $(B \cup F) \cap B' = \emptyset$, the symbols representing $\boldsymbol{b'}$ do not clash with the symbols representing $(\boldsymbol{f}, \boldsymbol{b})$, and therefore $(\boldsymbol{f}, \boldsymbol{b}, \boldsymbol{b'}) : \boldsymbol{s_{i'}}$ is well-defined and projects on the same indices as $(\boldsymbol{f'}, \boldsymbol{b'}) : \boldsymbol{s_{i'}}$. The fourth equality follows from the distributivity in a semiring. □

This means that we can compress all nested Einsum expressions, where the output string of the inner expression, which is used to compute U, is exactly the same as the respective input string in the outer expression, where U is used as an input tensor. This is already helpful for some naturally occuring expressions in linear algebra, e.g.

$$\begin{aligned} |A \cdot v|_2^2 &= (i, i \to, (ij, j \to i, A, v), (ij, j \to i, A, v)) \\ &= (ij, j, ij, j \to, A, v, A, v) \end{aligned}$$

for $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$, or

$$\begin{aligned} A \cdot B \cdot v &= (ij, j \to i, (ik, kj \to ij, A, B), v) \\ &= (ik, kj, j \to i, A, B, v) \end{aligned}$$

for $A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n}, v \in \mathbb{R}^n$. However, sometimes we need to access a different multi-index set than the one we computed, e.g.

$$\operatorname{trace}(A \cdot B) = (ii \to, (ik, kj \to ij, A, B))$$

or

$$A \cdot \operatorname{diag}(v) = (ik, kj \to ij, A, (i \to ii, v))$$

for $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times m}, v \in \mathbb{R}^n$. For this, we need more general ways of compressing nested Einsum expressions.

3.2 Introducing Duplications

The following example is another expression, which we cannot compress with the previous theorem:

$$(ij, jjj \to i, A, (kl, lo \to kko, B, C))$$

for $A \in \mathbb{R}^{a \times b}, B \in \mathbb{R}^{b \times c}, C \in \mathbb{R}^{c \times b}$. In the following, we will explore how to compress such expressions. Note that, for the theorem, we use disjoint sets of symbols for the inner and outer expression. This helps in the proof, and is not a real constraint in practice, because we can just rename the symbols in different scopes. For example, we could also write the above expression as

$$(ij, jjj \to i, A, (ik, kj \to iij, B, C)),$$

because the scope of each symbol does not reach into nested expressions, and therefore the j used in the outer expression and the j used in the inner expression are treated as different symbols.

Theorem 2: For $i \in [m+n]$, let $T^{(i)}$ be an n_i-th order tensor with index strings $\boldsymbol{s_i} \in S^{n_i}$. Let $\boldsymbol{s_u}$ be an index string for the n_u-th order tensor U, which is defined as follows:

$$U := (\boldsymbol{s_{m+1}}, \dots, \boldsymbol{s_{m+n}} \to \boldsymbol{s_u}, T^{(m+1)}, \dots, T^{(m+n)})$$

Also let $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ be alternative index strings for U with $s_{uj} = s_{uj'} \implies \hat{s}_{uj} = \hat{s}_{uj'}$ for all $j, j' \in [n_u]$, which means that $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ can only introduce new symbol duplications, and cannot remove any. The index string $\boldsymbol{s_u}$ corresponds to the output string of the inner expression, and the index string $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ corresponds to the input string that is used for the input tensor U in the outer expression.

In our example, $\boldsymbol{s_u} = kko$ and $\hat{\boldsymbol{s}}_{\boldsymbol{u}} = jjj$. This does not break the symbol duplication of the first and second index, and introduces a new duplication on the third index.

Let s_v be an index string and

$$V := (\boldsymbol{s_1}, \dots, \boldsymbol{s_m}, \hat{\boldsymbol{s}}_{\boldsymbol{u}} \to \boldsymbol{s_v}, T^{(1)}, \dots, T^{(m)}, U)$$

such that the first and second Einsum expression share no symbols. Then these nested Einsum expressions can also be compressed into a single Einsum expression.

In contrast to Theorem 1, we cannot just replace the input index string $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ by all the input index strings in the inner Einsum expression $\boldsymbol{s_{m+1}}, \dots, \boldsymbol{s_{m+n}}$. Instead, we first need to apply a symbol map to the input strings of the inner expression. Let $\nu : S \to S$ such that

$$\nu(s) := \begin{cases} \hat{s}_{uj} & \text{if } \exists j \in [n_u] : s_{uj} = s \\ s & \text{else} \end{cases}$$

which maps symbols in $\boldsymbol{s_u}$ to the symbol at the same index in $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ and all other symbols to themselves.

This symbol map holds information about which symbols will be iterated over at the same time in the outer expression. In our example, we have the following symbols on the same positions:

- $s_{u1} = k$ and $\hat{s}_{u1} = j$,
- $s_{u2} = k$ and $\hat{s}_{u2} = j$,
- $s_{u3} = o$ and $\hat{s}_{u3} = j$.

Therefore these are the important mappings:

$$k \to j,$$
$$o \to j.$$

This means that k and o will be iterated over at the same time.

The symbol map ν can be extended, such that it maps entire index strings instead of just symbols, by setting $\nu(\boldsymbol{s_i}) \in S^{n_i}, \nu(\boldsymbol{s_i})_j := \nu(s_{ij})$. Then we can write the substituted index strings by setting $\hat{\boldsymbol{s}}_{\boldsymbol{i}} := \nu(\boldsymbol{s_i})$ for $i \in [m+1, m+n]$.

The compressed Einsum expression now becomes the following:

$$V = (\boldsymbol{s_1}, \ldots, \boldsymbol{s_m}, \hat{\boldsymbol{s}}_{\boldsymbol{m+1}}, \ldots, \hat{\boldsymbol{s}}_{\boldsymbol{m+n}} \to \boldsymbol{s_v}, T^{(1)}, \ldots, T^{(m+n)})$$

which helps us to compress the example:

$$(ij, jjj \to i, A, (kl, lo \to kko, B, C)) = (ij, jl, lj \to i, A, B, C).$$

Proof. The fundamental idea behind this theorem is, that by using the index string $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$, we only iterate over a sub-space of the indices that we defined for the computation of U. The way in which we iterate over this sub-space is determined by the outer expression. It could either be the sum over bound indices or the universal quantifier over free indices. To formulate this, we need some idea of which multi-indices we iterate over. Therefore, let $\mathcal{I} : \boldsymbol{s} := \{\boldsymbol{i} : \boldsymbol{s} \mid \boldsymbol{i} \in \mathcal{I}\}$ for an index string $\boldsymbol{s}$ and a multi-index space $\mathcal{I}$.

Let F', B' be the free and bound symbols of the inner Einsum expression. W.l.o.g. they are both non-empty. From them we can derive the multi-index spaces $\mathcal{F}', \mathcal{B}'$ as in the definition. Let $\hat{F}' = \sigma(\hat{\boldsymbol{s}}_{\boldsymbol{u}})$ and $\hat{\mathcal{F}}' = \prod_{s \in \hat{F}'}[d_s]$. Then $\hat{\mathcal{F}}' : \hat{\boldsymbol{s}}_{\boldsymbol{u}} \subseteq \mathcal{F}' : \boldsymbol{s_u}$. This follows from $d_{s_{uj}} = d_{\hat{s}_{uj}}$ for $j \in [n_u]$, and the amount of symbols in the projection of $\hat{\mathcal{F}}' : \hat{\boldsymbol{s}}_{\boldsymbol{u}}$ being smaller or equal to the amount of symbols in the projection of $\mathcal{F}' : \boldsymbol{s_u}$. The first fact is true per the definition of Einsum. The second fact can be rewritten as $|\sigma(\hat{\boldsymbol{s}}_{\boldsymbol{u}})| \leq |\sigma(\boldsymbol{s_u})|$ and follows directly from the constraint $s_{uj} = s_{uj'} \implies \hat{s}_{uj} = \hat{s}_{uj'}$ for all $j, j' \in [n_u]$.

Then

$$\forall \boldsymbol{f'} \in \mathcal{F}' : U_{\boldsymbol{f'}:\boldsymbol{s_u}} = \bigoplus_{\boldsymbol{b'} \in \mathcal{B}'} \bigodot_{i=m+1}^{m+n} T^{(i)}_{(\boldsymbol{f'}, \boldsymbol{b'}):\boldsymbol{s_i}}$$

and therefore

$$\forall \hat{\boldsymbol{f}}' \in \hat{\mathcal{F}}' : U_{\hat{\boldsymbol{f}}':\hat{\boldsymbol{s}}_{\boldsymbol{u}}} = \bigoplus_{\boldsymbol{b'} \in \mathcal{B}'} \bigodot_{i=m+1}^{m+n} T^{(i)}_{(\hat{\boldsymbol{f}}', \boldsymbol{b'}):\hat{\boldsymbol{s}}_{\boldsymbol{i}}}$$

because of the previous observation, and because the bound symbols of the expression, which are used in $\boldsymbol{b'}$, do not occur in $\boldsymbol{s_u}$, and are therefore not changed by the symbol map ν. Therefore

$$U = (\hat{\boldsymbol{s}}_{\boldsymbol{m+1}}, \ldots, \hat{\boldsymbol{s}}_{\boldsymbol{m+n}} \to \hat{\boldsymbol{s}}_{\boldsymbol{u}}, T^{(m+1)}, \ldots, T^{(m+n)})$$

and we can use Theorem 1 for

$$V = (\boldsymbol{s_1}, \ldots, \boldsymbol{s_m}, \hat{\boldsymbol{s}}_{\boldsymbol{m+1}}, \ldots, \hat{\boldsymbol{s}}_{\boldsymbol{m+n}} \to \boldsymbol{s_v}, T^{(1)}, \ldots, T^{(m+n)})$$

because the bound symbols of the inner expression have not been mapped to any of the symbols used in the outer expression. □

With this theorem, we can prove a property of the trace in a relatively simple manner, namely that for $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times m}$, it holds that

$$\text{trace}(A \cdot B) = \text{trace}(B \cdot A).$$

Proof.

$$\begin{aligned}
\text{trace}(A \cdot B) &= (ll \to, (ik, kj \to ij, A, B)) \\
&= (lk, kl \to, A, B) \\
&= (kl, lk \to, B, A) \\
&= (kk \to, (il, lj \to ij, B, A)) \\
&= \text{trace}(B \cdot A)
\end{aligned}$$

where the second and fourth equality hold because of Theorem 2, and the third equality holds because of the commutativity of multiplication in the standard semiring. □

This is already a useful tool for compressing nested expressions, but there are still some naturally occuring expressions we cannot compress with this, e.g.:

$$A \cdot \text{diag}(v) = (ik, kj \to ij, A, (i \to ii, v))$$

for $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$. This is because the symbol duplication ii is broken by the index string kj, and therefore we access more entries than the ones we computed.

3.3 Removing Duplications

The following example is an expression, which we cannot compress with the previous theorems:

$$(ij, kl, mn, ijklmn \to ijk, A, B, C, (abc \to aabbcc, D))$$

for $A \in \mathbb{R}^{x \times x}, B \in \mathbb{R}^{y \times y}, C \in \mathbb{R}^{z \times z}, D \in \mathbb{R}^{x \times y \times z}$. This is because duplications in $\boldsymbol{s_u} = aabbcc$ are removed by the input string $\hat{\boldsymbol{s}}_{\boldsymbol{u}} = ijklmn$. In the following theorem, we will explore how to compress expressions such as this one. Again, we use disjoint sets of symbols for the inner and outer expression to help us in the formulation and the proof.

Theorem 3: For $i \in [m+n]$, let $T^{(i)}$ be an n_i-th order tensor with index strings $\boldsymbol{s_i} \in S^{n_i}$. Let $\boldsymbol{s_u}$ be an index string for the n_u-th order tensor U, which is defined as follows:

$$U := (\boldsymbol{s_{m+1}}, \ldots, \boldsymbol{s_{m+n}} \to \boldsymbol{s_u}, T^{(m+1)}, \ldots, T^{(m+n)})$$

Also let $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ be alternative index strings for U with $s_{uj} \neq s_{uj'} \implies \hat{s}_{uj} \neq \hat{s}_{uj'}$ for all $j, j' \in [n_u]$, which means that $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ can only remove symbol duplications, and cannot introduce any. Note that this is the converse of the constraint in Theorem 2.

In our example, $\boldsymbol{s_u} = oopp$ and $\hat{\boldsymbol{s}}_{\boldsymbol{u}} = jklm$. This removes the symbol duplication of the first and second index, as well as the symbol duplication of the third and fourth index.

Let s_v be an index string and

$$V := (\boldsymbol{s_1}, \ldots, \boldsymbol{s_m}, \hat{\boldsymbol{s}}_{\boldsymbol{u}} \to \boldsymbol{s_v}, T^{(1)}, \ldots, T^{(m)}, U)$$

where the first and second Einsum expression share no symbols. Then these nested Einsum expressions can also be compressed into a single Einsum expression.

As in Theorem 2, we need to apply a symbol map before substituting $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$. Interestingly, the symbol map is not applied to the index strings of the inner expression $(\boldsymbol{s_{m+1}}, \ldots, \boldsymbol{s_{m+n}})$, but to the index strings of the outer expression $(\boldsymbol{s_1}, \ldots, \boldsymbol{s_m}$ and $\boldsymbol{s_v})$. Similarly, it does not map $\boldsymbol{s_u}$ to $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$, but $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ to $\boldsymbol{s_u}$.

Let $\nu : S \to S$ such that

$$\nu(s) := \begin{cases} s_{uj} & \text{if } \exists j \in [n_u] : \hat{s}_{uj} = s \\ s & \text{else} \end{cases},$$

which can be extended to map entire index strings as in Theorem 2. In our example, these are the important mappings:

$$\begin{aligned} i &\to a, & k &\to b, & m &\to c, \\ j &\to a, & l &\to b, & n &\to c. \end{aligned}$$

This means that i and j will be iterated over at the same time, k and l will be iterated over at the same time, and m and l will be iterated over at the same time.

Let $\hat{\boldsymbol{s}}_{\boldsymbol{i}} := \nu(\boldsymbol{s_i})$ for $i \in [m]$, $\hat{\boldsymbol{s}}_{\boldsymbol{v}} := \nu(\boldsymbol{s_v})$, then the compressed Einsum expression is the following:

$$V = (\hat{\boldsymbol{s}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{s}}_{\boldsymbol{m}}, \boldsymbol{s_{m+1}}, \ldots, \boldsymbol{s_{m+n}} \to \hat{\boldsymbol{s}}_{\boldsymbol{v}}, T^{(1)}, \ldots, T^{(m+n)})$$

which helps us to compress the example:

$$\begin{aligned} &(ij, kl, mn, ijklmn \to ijk, A, B, C, (abc \to aabbcc, D)) \\ &\quad = (aa, bb, cc, abc \to aab, A, B, C, D). \end{aligned}$$

Note how the index string for the output $\boldsymbol{s_v}$ was changed into $\hat{\boldsymbol{s}}_{\boldsymbol{v}}$. This will become apparent in the proof.

Proof. The key idea behind this proof, is that the entries of U, which were not defined in the computation, are set to the additive neutral element $\mathbb{0}$. This is useful, because in a semiring over some set M, the additive neutral element *annihilates* M. This means, that for any $a \in M$, $a \cdot \mathbb{0} = \mathbb{0} \cdot a = \mathbb{0}$. Therefore, for any multi-index where U is set to $\mathbb{0}$, V is

also set to $\mathbb{0}$. This means, that in the computation of V, only the indices which respect the duplications in $\boldsymbol{s_u}$ are defined.

Let F, F', B, B' be the free and bound symbols of the outer and inner Einsum expression respectively. W.l.o.g. they are all non-empty. From them we can derive the multi-index spaces $\mathcal{F}, \mathcal{F}', \mathcal{B}, \mathcal{B}'$ as in the definition. Then $U_{(\boldsymbol{f},\boldsymbol{b}):\hat{\boldsymbol{s}}_{\boldsymbol{u}}}$ is only non-zero for multi-indices $(\boldsymbol{f}, \boldsymbol{b}) \in \mathcal{F} \times \mathcal{B}$ with $(\boldsymbol{f}, \boldsymbol{b}) : \hat{s}_{uj} = (\boldsymbol{f}, \boldsymbol{b}) : \hat{s}_{uj'}$, where $j, j' \in [n_u]$ are indices of $\boldsymbol{s_u}$ where the symbols are duplicated, i.e. $s_{uj} = s_{uj'}$. In our example, this means that $(op \to oopp, B)$ is only non-zero for $(j, k, l, m) \in [d_j] \times [d_k] \times [d_l] \times [d_m]$ with $j = k$ and $l = m$, because $s_{u1} = s_{u2} = o$ and $s_{u3} = s_{u4} = p$.

Therefore, when U is multiplied with the other tensors, the resulting entry

$$\bigodot_{i=1}^{m} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s_i}} \odot U_{(\boldsymbol{f},\boldsymbol{b}):\hat{\boldsymbol{s}}_{\boldsymbol{u}}}$$

is only non-zero for multi-indices $(\boldsymbol{f}, \boldsymbol{b}) \in \mathcal{F} \times \mathcal{B}$ that respect the same conditions. In our example, this is equivalent to

$$A_{ij}B_{kl}C_{mn} \odot U_{ijklmn} = \begin{cases} A_{ij}B_{kl}C_{mn} \odot U_{ijklmn} & \text{if } i = j, k = l, m = n \\ \mathbb{0} & \text{else} \end{cases}.$$

Now, this already looks like not all symbols are needed for this computation. But to see, in which way we can replace symbols, we need to consider the three ways in which duplications can be broken. Either a duplication is broken only by free symbols, only by bound symbols, or by a combination of both. In our example, we have all of these cases. The duplication aa is broken by i and j, which are both free symbols. The duplication cc is broken by m and n, which are both bound symbols. The duplication bb is broken by k and l, where k is a free symbol and l is a bound symbol. Every one of these cases leads to the same result, but in a slightly different way.

First let us consider the case where a duplication is broken only by free symbols. In this case, the free symbols that break the duplication can be replaced by a single symbol, because all entries of V, with a multi-index that does not respect the duplication, is $\mathbb{0}$. In our example, this is equivalent to replacing i and j by a single symbol a:

$$\begin{aligned} \forall i, j, k : V_{ijk} &= \bigoplus_{l,m,n} A_{ij}B_{kl}C_{mn} \odot U_{ijklmn} \\ \iff \forall a, k : V_{aak} &= \bigoplus_{l,m,n} A_{aa}B_{kl}C_{mn} \odot U_{aaklmn} \end{aligned}$$

For the next two cases, we need to use that $a \oplus \mathbb{0} = a$ for any $a \in M$. This means, that only those summands that respect the duplications will be summed over, because all summands which do not respect the summation are $\mathbb{0}$. This affects the remaining two cases in different ways. If a duplication is broken only by bound symbols, then we need a single symbol to sum over all the multi-indices that respect the duplication. In our example, this is equivalent to replacing m and n by a single symbol c:

$$\bigoplus_{l,m,n} A_{aa}B_{kl}C_{mn} \odot U_{aaklmn} = \bigoplus_{l,c} A_{aa}B_{kl}C_{cc} \odot U_{aaklcc}$$

Now, if a duplication is broken by free symbols and by bound symbols, then the free symbols can again be replaced by a single symbol, as in the first case. In our example, this is useless, because there is already only one symbol k where this can be applied. Let us do it anyway, because it might clear up what needs to be done in this step.

$$\begin{aligned} &\forall a,k : V_{aak} = \bigoplus_{l,c} A_{aa}B_{kl}C_{cc} \odot U_{aaklcc} \\ \iff &\forall a,b : V_{aab} = \bigoplus_{l,c} A_{aa}B_{bl}C_{cc} \odot U_{aablcc} \end{aligned}$$

Then, the values held by the breaking bound symbols are already defined by the value held by the now only breaking free symbol. Therefore, the summation over the values of this symbol is useless, because there is only one combination of indices, which respects the duplication. Therefore, the breaking bound symbols need to hold the same exact value as the breaking free symbol, and these symbols can be replaced by the same symbol, that was used to replace the free symbols. Because of this, the occurence of the symbol also needs to be removed from the sum. In our example, this is equivalent to replacing l by b and removing it from the sum:

$$\begin{aligned} \forall a,b : V_{aab} &= \bigoplus_{l,c} A_{aa}B_{bl}C_{cc} \odot U_{aablcc} \\ &= \bigoplus_{c} A_{aa}B_{bb}C_{cc} \odot U_{aabbcc} \end{aligned}$$

Therefore, in all three cases, the symbols, that break a duplication, can simply be replaced by a single symbol. Conveniently, as a replacing symbol, we can just use the symbol that defined the duplication in the first place, because the inner expression shares no symbols with the outer expression. This yields exactly the symbol map ν. Therefore $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$, which is the index string of U that was used in the outer expression, will be replaced by $\boldsymbol{s}_{\boldsymbol{u}}$, which is the index string of U that was used in the inner expression. Just as convenient is, that replacing the breaking symbols that include a combination of free and bound symbols, already removes the breaking bound symbols from the sum, because the bound symbols are by definition only those symbols, which are not free. In our example, this means that replacing k and l by b already removes l from the sum, because b is now a free symbol as well.

For the final steps, we need to define new multi-index sets to iterate and sum over. For this, let $\hat{F} := \sigma(\hat{\boldsymbol{s}}_{\boldsymbol{v}})$ and $\hat{B} := \left(\bigcup_{i\in[m]} \sigma(\hat{\boldsymbol{s}}_{\boldsymbol{i}}) \cup \boldsymbol{s}_{\boldsymbol{u}}\right) \setminus \sigma(\hat{\boldsymbol{s}}_{\boldsymbol{v}})$. Let $\hat{\mathcal{F}} = \prod_{s\in\hat{F}}[d_s]$ and $\hat{\mathcal{B}} = \prod_{s\in\hat{B}}[d_s]$. Then

$$\begin{aligned} &V = (\boldsymbol{s}_{\boldsymbol{1}}, \dots, \boldsymbol{s}_{\boldsymbol{m}}, \hat{\boldsymbol{s}}_{\boldsymbol{u}} \to \boldsymbol{s}_{\boldsymbol{v}}, T^{(1)}, \dots, T^{(m)}, U) \\ \iff &\forall \boldsymbol{f} \in \mathcal{F} : V_{\boldsymbol{f}:\boldsymbol{s}_{\boldsymbol{v}}} = \bigoplus_{\boldsymbol{b}\in\mathcal{B}} \bigodot_{i=1}^{m} T^{(i)}_{(\boldsymbol{f},\boldsymbol{b}):\boldsymbol{s}_{\boldsymbol{i}}} \odot U_{(\boldsymbol{f},\boldsymbol{b}):\hat{\boldsymbol{s}}_{\boldsymbol{u}}} \\ \iff &\forall \hat{\boldsymbol{f}} \in \hat{\mathcal{F}} : V_{\hat{\boldsymbol{f}}:\hat{\boldsymbol{s}}_{\boldsymbol{v}}} = \bigoplus_{\hat{\boldsymbol{b}}\in\hat{\mathcal{B}}} \bigodot_{i=1}^{m} T^{(i)}_{(\hat{\boldsymbol{f}},\hat{\boldsymbol{b}}):\hat{\boldsymbol{s}}_{\boldsymbol{i}}} \odot U_{(\hat{\boldsymbol{f}},\hat{\boldsymbol{b}}):\boldsymbol{s}_{\boldsymbol{u}}} \\ \iff &V = (\hat{\boldsymbol{s}}_{\boldsymbol{1}}, \dots, \hat{\boldsymbol{s}}_{\boldsymbol{m}}, \boldsymbol{s}_{\boldsymbol{u}} \to \hat{\boldsymbol{s}}_{\boldsymbol{v}}, T^{(1)}, \dots, T^{(m)}, U) \end{aligned}$$

where the second equivalence holds because of the previously discussed symbol replacements. Then we can use Theorem 1 for

$$V = (\hat{\boldsymbol{s}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{s}}_{\boldsymbol{m}}, \boldsymbol{s}_{\boldsymbol{m+1}}, \ldots, \boldsymbol{s}_{\boldsymbol{m+n}} \to \hat{\boldsymbol{s}}_{\boldsymbol{v}}, T^{(1)}, \ldots, T^{(m+n)})$$

because the symbols of the outer expression have not been mapped to any of the bound symbols of the inner expression. □

With these theorems, we can write every naturally occuring complex expression from linear algebra as a single Einsum expression. The reason for this is, that in linear algebra, only up to two indices are used for a single tensor, which means that with two index strings, it cannot happen that a duplication is removed and anonther duplication is introduced simultaniously. Here are some more complex expressions as examples:

- squared norm of matrix-vector multiplication: Let $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$. Then

$$\begin{aligned} |A \cdot v|_2^2 &= (i, i \to, (ij, j \to i, A, v), (ij, j \to i, A, v)) \\ &= (ij, j, ij, j \to, A, v, A, v) \end{aligned}$$

- trace of matrix-matrix multiplication: Let $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times m}$. Then

$$\begin{aligned} \text{trace}(A \cdot B) &= (ii \to, (ik, kj \to ij, A, B)) \\ &= (ik, ki \to, A, B) \end{aligned}$$

- matrix multiplication with a diagonal matrix: Let $A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^n$. Then

$$\begin{aligned} A \cdot \text{diag}(v) &= (ik, kj \to ij, A, (i \to ii, v)) \\ &= (ij, j \to ij, A, v) \end{aligned}$$

But to write every expression from linear algebra as a single Einsum expression respectively was already possible before, with (TODO: cite Julien). With these theorems, we just derived a different way of achieving that. For this, we can state a very simple procedure. First, every function is translated to their respective Einsum expression, which results in a nested Einsum expression. Then, the nested expressions are compressed from the bottom up.

But we still do not have a way of compressing general nested Einsum expressions, where duplications might be removed and introduced simultaniously. If we could do that, then we could build a compiler that is able to compress every possible nested Einsum expression, regardless of duplications.

3.4 General Nested Expressions

In the final generalisation of compressing nested Einsum expressions, all duplication breaking is allowed. This has no application for linear algebra, as all the previous theorems had, because this first comes into play with third order tensors. But it still serves as a useful tool, because with it, we are able to compress every nested Einsum expression, if the nested expressions are over the same semiring.

The following example is an expression, which we cannot compress with the previous theorems:

$$(a, b, c, d, e, abbde \to bc, v^{(1)}, v^{(2)}, v^{(3)}, v^{(4)}, v^{(5)}, (i, j, k, l \to iijkkl, v^{(6)}, v^{(7)}, v^{(8)}, v^{(9)}))$$

for $v^{(i)} \in \mathbb{R}^{d_{vi}}$ with $i \in [9]$ and

$$\begin{aligned} d_{v1} = d_{v2} = d_{v6} = d_{v7}, \\ d_{v2} = d_{v3} = d_{v8}, \\ d_{v5} = d_{v9}. \end{aligned}$$

This is because, from the output string $\boldsymbol{s_u} = iijkkl$ to the input string $\hat{\boldsymbol{s}}_{\boldsymbol{u}} = abbde$, duplications are simultaniously removed and introduced. For example, the duplication ii is removed by the symbols ab, and a new duplication bb is introduced on the second and third index, which holds the symbols ij in the output string. In the following theorem, we will explore how to compress such expressions. Again, we use disjoint sets of symbols for the inner and outer expression to help us in the formulation.

Conjecture 4: For $i \in [m+n]$, let $T^{(i)}$ be an n_i-th order tensor with index strings $\boldsymbol{s_i} \in S^{n_i}$. Let $\boldsymbol{s_u}$ be an index string for the n_u-th order tensor U, which is defined as follows:

$$U := (\boldsymbol{s_{m+1}}, \dots, \boldsymbol{s_{m+n}} \to \boldsymbol{s_u}, T^{(m+1)}, \dots, T^{(m+n)})$$

Also let $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ be alternative index strings for U.

Let s_v be an index string and

$$V := (\boldsymbol{s_1}, \dots, \boldsymbol{s_m}, \hat{\boldsymbol{s}}_{\boldsymbol{u}} \to \boldsymbol{s_v}, T^{(1)}, \dots, T^{(m)}, U)$$

where the first and second Einsum expression share no symbols. Then these nested Einsum expressions can also be compressed into a single Einsum expression.

Once again, a map $\nu : S \to S$ has to be applied to the index strings before substituting index strings. The definition of the map this time is somewhat more complex. As in the previous two theorems, this map holds information about which symbols are essentially used together as one index.

For the definition of the map ν, we first construct an undirected graph $G = (V, E)$ that we call *symbol graph*. In the symbol graph, the nodes consist of all symbols from both expressions. The edges are $E = \{\{s_{uj}, \hat{s}_{uj}\} \mid j \in [n_u]\}$, which connects all

symbols from $\boldsymbol{s_u}$ and $\hat{\boldsymbol{s}}_{\boldsymbol{u}}$ that share an index. The symbol graph for our example is displayed in Figure 3.1.

In the symbol graph, if two symbols are connected, then they need to be iterated over at the same time in the compressed expression, because they are essentially the same index. Therefore, it makes sense assigning a symbol $s_C \in S \setminus V$ to each of the graphs components C. Then we can define ν as follows:

$$\nu(s) := \begin{cases} s_C & \text{if } s \in C \\ s & \text{else} \end{cases},$$

which can be extended to map entire index strings as in Theorem 2. In our example, the components are $\{a, b, i, j\}$, $\{c, d, k\}$, and $\{e, l\}$. Therefore we could use

$$\nu(s) := \begin{cases} x & \text{if } s \in \{a, b, i, j\} \\ y & \text{if } s \in \{c, d, k\} \\ z & \text{if } s \in \{e, l\} \\ s & \text{else} \end{cases}.$$

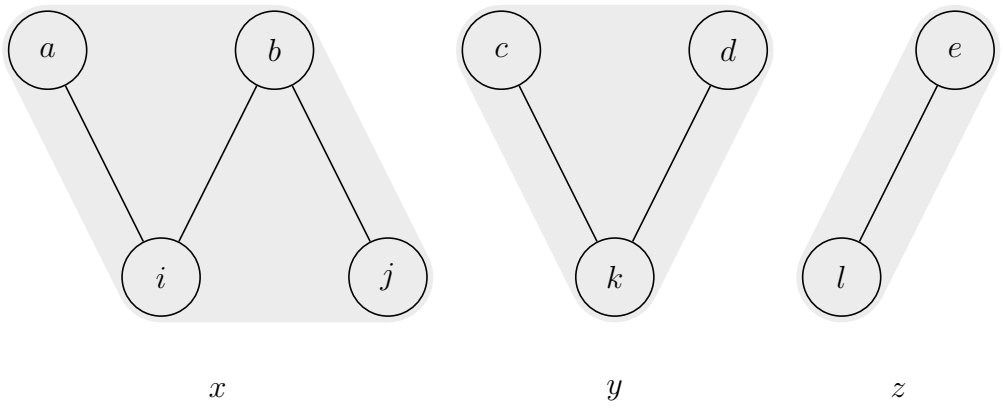


Figure 3.1: Symbol graph for the example

In this general form, this map is applied to all index strings from both expressions before the substitution. Let $\hat{\boldsymbol{s}}_{\boldsymbol{i}} := \nu(\boldsymbol{s_i})$ for $i \in [m+n]$, $\hat{\boldsymbol{s}}_{\boldsymbol{v}} := \nu(\boldsymbol{s_v})$, then the compressed Einsum expression is the following:

$$V = (\hat{\boldsymbol{s}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{s}}_{\boldsymbol{m+n}} \to \hat{\boldsymbol{s}}_{\boldsymbol{v}}, T^{(1)}, \ldots, T^{(m+n)})$$

which helps us to compress the example:

$$\begin{aligned}(a, b, c, d, e, abbde \to bc, v^{(1)}, v^{(2)}, v^{(3)}, v^{(4)}, v^{(5)}, (i, j, k, l \to iijkkl, v^{(6)}, v^{(7)}, v^{(8)}, v^{(9)})) \\ = (x, x, y, y, z, x, x, y, z \to xy, v^{(1)}, v^{(2)}, v^{(3)}, v^{(4)}, v^{(5)}, v^{(6)}, v^{(7)}, v^{(8)}, v^{(9)}).\end{aligned}$$

4 Naturally Occuring Einsum Expressions

4.1 Discrete Fourier Transform

Let T be an n-th order tensor where all axes have size m. Then the Discrete Fourier Transform $\mathrm{DFT}(T)$ is the n-th order tensor where all axes have size m with:

$$\mathrm{DFT}(T)_{x_1,\ldots,x_n} = \sum_{y_1,\ldots,y_n \in [m]} T_{y_1,\ldots,y_n} \cdot \prod_{j+k \leq n+1} \exp\left(-i2\pi \frac{(x_j - 1)(y_k - 1)}{m^{n-j-k+2}}\right)$$

for $x_1, \ldots, x_n \in [m]$. This formulation of the DFT is slightly rewritten form of the formulation by Aji [1].

To write the DFT as an Einsum expression, we need to define $\binom{n+1}{2}$ matrices $T^{(j,k)} \in \mathbb{C}^{m \times m}$ for $j + k \leq n + 1$ in the following way:

$$T^{(j,k)}_{x_j y_k} := \exp\left(-i2\pi \frac{(x_j - 1)(y_k - 1)}{m^{n-j-k+2}}\right)$$

for $x_j, y_k \in [m]$. Then

$$\mathrm{DFT}(T)_{x_1,\ldots,x_n} = \sum_{y_1,\ldots,y_n \in [m]} T_{y_1,\ldots,y_n} \cdot \prod_{j+k \leq n+1} T^{(j,k)}_{x_j y_k}$$

for $x_1, \ldots, x_n \in [m]$. Therefore the DFT can be written as an Einsum expression with index strings consisting of symbols s_{xi} and s_{yj} for $j, k \in [n]$:

- $\boldsymbol{s_x} = s_{x1} \ldots s_{xn}$
- $\boldsymbol{s_y} = s_{y1} \ldots s_{yn}$
- $\boldsymbol{s_{j,k}} = s_{xj} s_{yk}$ for $j, k \in [n]$

With these index strings, the Einsum expression for a general DFT over any number of dimensions is the following:

$$\begin{aligned}\mathrm{DFT}(T) = (&\boldsymbol{s_y}, \boldsymbol{s_{1,1}}, \ldots, \boldsymbol{s_{1,n}}, \boldsymbol{s_{2,1}}, \ldots, \boldsymbol{s_{2,n-1}}, \ldots, \boldsymbol{s_{n,1}} \to \boldsymbol{s_x}, \\ &T, T^{(1,1)}, \ldots, T^{(1,n)}, T^{(2,1)}, \ldots, T^{(2,n-1)}, \ldots, T^{(n,1)})\end{aligned}$$

Because this notation is hard to read, we will explore an example of the DFT of a third-order tensor with axes of size 32. In this example, we have to define 6 matrices $T^{(1,1)}$, $T^{(1,2)}$, $T^{(1,3)}$, $T^{(2,1)}$, $T^{(2,2)}$, and $T^{(3,1)}$. We will use a, b, c as symbols for the indices of T, and x, y, z as the symbols for the indices of $\mathrm{DFT}(T)$. Then the matrices are defined in the following way:

$$T^{(1,1)}_{xa} = \exp\left(-i2\pi\frac{(x-1)(a-1)}{32^3}\right) \qquad T^{(1,3)}_{xc} = \exp\left(-i2\pi\frac{(x-1)(c-1)}{32}\right)$$
$$T^{(1,2)}_{xb} = \exp\left(-i2\pi\frac{(x-1)(b-1)}{32^2}\right) \qquad T^{(2,2)}_{yb} = \exp\left(-i2\pi\frac{(y-1)(b-1)}{32}\right)$$
$$T^{(2,1)}_{ya} = \exp\left(-i2\pi\frac{(y-1)(a-1)}{32^2}\right) \qquad T^{(3,1)}_{za} = \exp\left(-i2\pi\frac{(z-1)(a-1)}{32}\right)$$

for $x, y, z, a, b, c \in [32]$. Then the Einsum expression is the following:

$$\begin{aligned}\mathrm{DFT}(T) = (&abc, xa, xb, xc, ya, yb, za \to xyz,\\ &T, T^{(1,1)}, T^{(1,2)}, T^{(1,2)}, T^{(2,1)}, T^{(2,2)}, T^{(3,1)})\end{aligned}$$

The inverse transform is anologue with

$$\hat{T}^{(j,k)}_{x_j y_k} := \exp\left(i2\pi\frac{(x_j-1)(y_k-1)}{m^{n-j-k+2}}\right).$$

4.2 Hadamard Transform

Let T be an n-th order tensor where all axes have size m. Then the Hadamard Transform $H(T)$ is the n-th order tensor where all axes have size m with:

$$H(T)_{x_1\dots x_n} = \sum_{y_1,\dots,y_n\in[m]} T_{y_1\dots y_n}(-1)^{x_1y_1+\dots+x_ny_n}$$

for $x_1, \dots, x_n \in [m]$.

Then we can define n matrices $T^{(i)} \in \mathbb{R}^{m\times m}$ for $i \in n$:

$$T^{(i)}_{x_iy_i} = (-1)^{x_iy_i}$$

for $x_i, y_i \in [m]$. Then

$$H(T)_{x_1\dots x_n} = \sum_{y_1\dots y_n\in[m]} T_{y_1\dots y_n} \cdot \prod_{i\in[n]} T^{(i)}_{x_iy_i}$$

for $x_1, \dots, x_n \in [m]$. Therefore the Hadamard Transform can be written as an Einsum expression with index strings consisting of symbols s_{xi} and s_{yj} for $j, k \in [n]$:

- $\boldsymbol{s_x} = s_{x1} \dots s_{xn}$
- $\boldsymbol{s_y} = s_{y1} \dots s_{yn}$
- $\boldsymbol{s_i} = s_{xi} s_{yi}$ for $i \in [n]$

Then

$$H(T) = (\boldsymbol{s_y}, \boldsymbol{s_1}, \dots, \boldsymbol{s_n} \to \boldsymbol{s_x}, T, T^{(1)}, \dots, T^{(n)}).$$

5 Deep Learning

5.1 Fully Connected Feed-Forward Net

A single layer of a fully connected Feed-Forward Neural Net with ReLU activations can be expressed as a nested Einsum expression, with the use of multiple semirings. For this, let

- $R_{(+,\cdot)}$ be the standard semiring $(\mathbb{R}, +, \cdot)$,
- $R_{(\max,+)}$ be the tropical semiring $(\mathbb{R} \cup \{-\infty\}, \max, +)$,
- $R_{(\min,\max)}$ be the minimax semiring $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, \max)$.

Let $\nu : \mathbb{R}^n \to \mathbb{R}^m$ be the function that computes the output of the layer for a given input vector $x \in \mathbb{R}^n$, with weights $A \in \mathbb{R}^{m \times n}$ and biases $b \in \mathbb{R}^m$. Then

$$\begin{aligned}\nu(x) &= \max(Ax + b, 0) \\ &= (i, i \to i, 0, (i, i \to i, b, (ij, j \to i, A, x)_{R_{(+,\cdot)}})_{R_{(\max,+)}})_{R_{(\min,\max)}}.\end{aligned}$$

A multi-layer network can the be achieved by nesting the respective Einsum expressions of each layer.

Because each level of nesting needs a different semiring, we can not use the theorems from Chapter 3 to compress the expression. But if we could find a way of compressing the massively nested expressions of deeper neural networks despite of that, then we could benefit from the advantages mentioned in Chapter 2 such as the optimisation of contraction paths. Unfortunately, it is unlikely that this is possible, because we found that expanding the matrix multiplication, that transforms the outputs of another layer, results in an exponentially big term.

To see this, we use the tropical semiring $(\mathbb{R}, \oplus, \odot)$, where $a \oplus b = \max(a, b)$ and $a \odot b = a + b$. We do this because tropical semiring can naturally express all the operations used in a fully connected neural network. For this we need to define the tropical power:

$$a^{\odot n} = \underbrace{a \odot a \odot \ldots \odot a}_{n \text{ times}}$$

for $n \in \mathbb{N}$. The following property of the tropical power is also needed:

$$\left(a^{\odot b}\right)^{\odot c} = a^{\odot(b+c)}.$$

The tropical semiring also allows us to use the distributive law of maximization and addition

$$a \odot (b \oplus c) = a \odot b \oplus a \odot c,$$

as well as the distributive law of addition and multiplication

$$(a \odot b)^{\odot n} = a^{\odot n} \odot b^{\odot n},$$

and the distributive law of maximization and multiplication, which is restricted on natural numbers

$$(a \oplus b)^{\odot n} = a^{\odot n} \oplus b^{\odot n}.$$

Let $\nu : \mathbb{R}^n \to \mathbb{R}^l$ be the function that computes the output of a two-layer fully connected neural network ($n \to m \to l$ neurons) with ReLU activations, which maps inputs $x \in \mathbb{R}^n$ to outputs $\nu(y) \in R^l$, with parameters $A^{(0)} \in \mathbb{R}^{m \times n}, A^{(1)} \in \mathbb{R}^{l \times m}, b^{(0)} \in \mathbb{R}^m, b^{(1)} \in \mathbb{R}^l$. Then the computation of the neural network is:

$$\nu(x) = \max(A^{(1)} \max(A^{(0)} x + b^{(0)}, 0) + b^{(1)}, 0)$$

In order to reasonably work with matrix multiplication in the tropcial semiring, we can only view matrices with positive integer entries. This is not a limitation, because making the entries integers does not impact the strength of the neural network [see 2, sec. 4].

In order to only use positive valued matrices, we can rewrite the expression of computing the next layer from a previous layer:

$$\begin{aligned} \max(Ax + b, 0) &= \max(A_+ x - A_- x + b, A_- x - A_- x) \\ &= \max(A_+ x + b, A_- x) - A_- x \end{aligned}$$

where $A_+ = \max(A, 0), A_- = \max(-A, 0)$ and therefore $A = A_+ - A_-$. This turns the network output into a tropical rational function [see 2, sec. 5]:

$$\begin{aligned} \nu(x) = \max(&\overbrace{A_+^{(1)} \max(A_+^{(0)} x + b^{(0)}, A_-^{(0)} x)}^{z} + A_-^{(1)} A_+^{(0)} x + b^{(1)}, \\ &A_-^{(1)} \max(A_+^{(0)} x + b^{(0)}, A_-^{(0)} x) + A_+^{(1)} A_+^{(0)} x) \\ &- \left[A_-^{(1)} \max(A_+^{(0)} x + b^{(0)}, A_-^{(0)} x) + A_+^{(1)} A_+^{(0)} x \right] \end{aligned}$$

We focus on the subexpression z, which makes the calculation a bit simpler, but keeps the point.

Now, if we want to avoid switching semirings, we need to apply the distributive law

multiple times.

$$
\begin{aligned}
z &= A_+^{(1)} \max(A_+^{(0)} x + b^{(0)}, A_-^{(0)} x) \\
z_i &= \bigodot_{j=1}^{m} \left(b_j^{(0)} \odot \bigodot_{k=1}^{n} x_k^{\odot A_{jk+}^{(0)}} \oplus \bigodot_{k=1}^{n} x_k^{\odot A_{jk-}^{(0)}} \right)^{\odot A_{ij+}^{(1)}} \\
&= \bigodot_{j=1}^{m} \left(\left(b_j^{(0)} \right)^{\odot A_{ij+}^{(1)}} \odot \bigodot_{k=1}^{n} x_k^{\odot \left(A_{ij+}^{(1)} + A_{jk+}^{(0)} \right)} \oplus \bigodot_{k=1}^{n} x_k^{\odot \left(A_{ij+}^{(1)} + A_{jk-}^{(0)} \right)} \right) \\
&= \bigoplus_{J \in 2^{[m]}} \bigodot_{j \in J} \left[\left(b_j^{(0)} \right)^{\odot A_{ij+}^{(1)}} \odot \bigodot_{k=1}^{n} x_k^{\odot \left(A_{ij+}^{(1)} + A_{jk+}^{(0)} \right)} \right] \odot \bigodot_{j \in [n] \setminus J} \left[\bigodot_{k=1}^{n} x_k^{\odot \left(A_{ij+}^{(1)} + A_{jk-}^{(0)} \right)} \right]
\end{aligned}
$$

Where the second equality is just the first equality written with the operations of the tropical semiring, the third equality follows from the distributive law of standard addition and multiplication and the distributive law of maximization and multiplication, and the last equality follows from the distributive law of maximization and addition.

This expression maximizes over a number of subexpressions that grows exponentially in the width of the inner layer. Which subexpressions can be removed before the evaluation remains an open question. Note that it depends on the non-linearities of the neural network, which might make it hard to find a general answer to this question.

5.2 Attention

For $Q \in \mathbb{R}^{d_v \times d_k}, K \in \mathbb{R}^{d_v \times d_k}, V \in \mathbb{R}^{d_v \times d_v}$, the attention mechanism is the following:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_K}} \right) V.$$

It is comprised of multiple steps, which can all be expressed with Einsum expressions and element-wise operations:

- matrix multiplication QK^T:

$$
\begin{aligned}
(QK^\top)_{ij} &= \sum_{k \in [d_k]} Q_{ik} K_{jk} \\
QK^\top &= (ik, jk \to ij, Q, K)
\end{aligned}
$$

- scaling by $\sqrt{d_k}$ (no Einsum needed)
- normalizing with softmax: Let $X \in \mathbb{R}^{m \times n}$, then

$$\text{softmax}(X)_{ij} := \frac{\exp(X_{ij})}{\omega_i}$$

where

$$\omega_i := \sum_{j \in [n]} \exp(X_{ij}).$$

Therefore

$$\text{softmax}(X) = (ij, i \to ij, \exp(X), 1/(ij \to i, \exp(X)))$$

- another matrix multiplication with V. Let $X \in \mathbb{R}^{d_v \times d_v}$:

$$XV = (ik, kj \to ij, X, V)$$

Then the whole attention mechanism can be expressed with Einsum expressions and the use of element-wise operations:

$$\begin{aligned}
\text{Attention}(Q, K, V) &= \text{softmax}\left(\frac{QK^\top}{\sqrt{d_K}}\right) V \\
&= (ik, kj \to ij, (ij, i \to ij, \exp(\frac{1}{\sqrt{d_k}} \cdot (ik, jk \to ij, Q, K)), \\
&\quad 1/(ij \to i, \exp(\frac{1}{\sqrt{d_k}} \cdot (ik, jk \to ij, Q, K)))), V) \\
&= (ik, kj, k \to ij, \exp(\frac{1}{\sqrt{d_k}} \cdot (ik, jk \to ij, Q, K)) \\
&\quad 1/(ij \to i, \exp(\frac{1}{\sqrt{d_k}} \cdot (ik, jk \to ij, Q, K))), V)
\end{aligned}$$

5.3 Batch Norm

Let $x^{(1)}, \ldots, x^{(m)}$ be m 4th-order tensors. Let X be the 5th-order tensor that consists of all those tensors combined along a new axis. Then the Batch Norm with parameters $\gamma, \beta \in \mathbb{R}$ over this *mini-batch* of tensors is defined in the following way:

$$\text{BN}_{\gamma,\beta}(X)_i = \frac{x_i - \text{E}_j[x_j]}{\sqrt{\text{Var}_j[x_j] + \epsilon}} \cdot \gamma + \beta$$

for $i \in [m]$, where ϵ is some constant added for numerical stability [3]. This computation is comprised of multiple steps, which can all be expressed with Einsum expressions and element-wise operations:

- mini-batch mean:

$$\mu_{abcd} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}_{abcd}$$

$$\mu = \frac{1}{m}(abcdi \to abcd, X)$$

- centralize:

$$\begin{aligned}\overline{x}^{(i)}_{abcd} &= x^{(i)}_{abcd} - \mu_{abcd} \\ \overline{X} &= (abcdi, abcd \to abcd, X, -\mu)_{R_{(\max,+)}}\end{aligned}$$

- mini-batch variance:

$$\begin{aligned}\sigma^2_{abcd} &= \frac{1}{m}\sum_{i=1}^{m}\left(\overline{x}^{(i)}_{abcd}\right)^2 \\ \sigma^2 &= \frac{1}{m}(abcdi, abcdi \to abcd, \overline{X})\end{aligned}$$

- normalize:

$$\begin{aligned}\hat{x}^{(i)}_{abcd} &= \frac{\overline{x}^{(i)}_{abcd}}{\sqrt{\sigma^2_{abcd} + \epsilon}} \\ \hat{X} &= (abcdi, abcd \to abcd, \overline{X}, (\sigma^2 + \epsilon)^{-\frac{1}{2}})\end{aligned}$$

- scale and shift:

$$\begin{aligned}y^{(i)}_{abcd} &= \gamma \hat{x}^{(i)}_{abcd} + \beta \\ Y &= \gamma \hat{X} + \beta\end{aligned}$$

Therefore the whole attention mechanism can be expressed with Einsum expressions and the use of element-wise operations:

$$\begin{aligned}\mathrm{BN}_{\gamma,\beta}(X) = (&abcdi, abcd \to abcd, \\ &(abcdi, abcd \to abcd, X, -\frac{1}{m}(abcdi \to abcd, X))_{R_{(\max,+)}}, \\ &(\frac{1}{m}(abcdi, abcdi \to abcd, \\ &\quad (abcdi, abcd \to abcd, X, -\frac{1}{m}(abcdi \to abcd, X))_{R_{(\max,+)}} \\ &)+\epsilon)^{-\frac{1}{2}} \\ ) &\cdot \gamma + \beta\end{aligned}$$

5.4 Convolution

Let F and G be two n-th order tensors where all axes of F have size d_F and all axes of G have size d_G with $d_F < d_G$. Let $d_O = d_G - d_F + 1$. Then the convolution $F * G$

is defined in the following way:

$$(F * G)_x := \sum_{y \in [d_F]^n} F_y \cdot G_{x+d_F-y}$$

for all $x \in [d_O]^n$, where $x+d_F-y$ indicates the element-wise addition $(x+d_F-y)_i = x_i + d_F - y_i$ for $i \in [n]$. Let

$$G'_{(x,y)} := G_{x+d_F-y}$$

for $x \in [d_O]^n, y \in [d_F]^n$. Then

$$(F * G)_x = \sum_{y \in [d_G]^n} F_y G'_{(x,y)}$$

for $x \in [d_O]^n$. Let

$$P_{(x,y,z)} := \begin{cases} 1 & \text{if } z = x + d_F - y \\ 0 & \text{else} \end{cases}$$

for $x \in [d_O]^n, y \in [d_F]^n, z \in [d_G]^n$. Then

$$G'_{(x,y)} = \sum_{z \in [d_G]^n} P_{(x,y,z)} G_z$$

for $x \in [d_O]^n, y \in [d_F]^n$. Therefore, convolution can be expressed as an Einsum expression:

$$(F * G) = ((\boldsymbol{s_x}, \boldsymbol{s_y}, \boldsymbol{s_z}), \boldsymbol{s_y}, \boldsymbol{s_z} \to \boldsymbol{s_x}, P, F, G)$$

where $\boldsymbol{s_x}, \boldsymbol{s_y}, \boldsymbol{s_z} \in S^n$ use distinct symbols.

The manual computation of the design tensor[1] P is quite expensive, and therefore this expression could be inefficient. It could lead to a more efficient computation, if this design tensor could be expressed as an *outer product* of 2 or more smaller tensors. Sadly, this is not possible.

Proof. To prove this, we will first show that, if P can be expressed as an outer product, then the factors also have to be scaled design tensors. Then we will give an example of a convolution, where P can not be expressed as an outer product of design tensors.

Let U be an m-th order tensor, and V an $(3n-m)$-th order tensor for $1 \leq m < 3n$. Let $\boldsymbol{s_u} \in S^m$ be the index string for U, and let $\boldsymbol{s_v} \in S^{3n-m}$ be the index string for V such that $\boldsymbol{s_u}$ and $\boldsymbol{s_v}$ use distinct symbols. Then P being the outer product of U and V means

$$P = (\boldsymbol{s_u}, \boldsymbol{s_v} \to (\boldsymbol{s_u}, \boldsymbol{s_v}), U, V)$$

up to reordering of axes.

Let u and v be entries of U and V respectively, and let $\boldsymbol{i_u}$ and $\boldsymbol{i_v}$ be a multi-index of U and V respectively, where this value occurs. Then the value $u \cdot v$ occurs in P at the multi-index

[1] A design tensor is a tensor that is filled only with the additive and multiplicative neutral element of the used semiring.

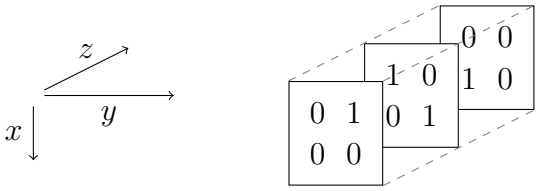


Figure 5.1: P for $F \in \mathbb{R}^2$ and $G \in \mathbb{R}^3$

$(\boldsymbol{i_u}, \boldsymbol{i_v})$. Therefore, if the sets of values contained in U and V are not of the form $\{0, c\}$ and $\{0, \frac{1}{c}\}$ for some $c \in \mathbb{R}$, then we can produce more values than 0 and 1 in P. Therefore U and V must be design tensors that were scaled by c and $\frac{1}{c}$ respectively for some $c \in \mathbb{R}$.

W.l.o.g. we now assume that U and V are both unscaled design tensors, because if

$$P = (\boldsymbol{s_u}, \boldsymbol{s_v} \to (\boldsymbol{s_u}, \boldsymbol{s_v}), cU, \frac{1}{c}V)$$

then

$$\begin{aligned} P &= c \cdot \frac{1}{c} \cdot (\boldsymbol{s_u}, \boldsymbol{s_v} \to (\boldsymbol{s_u}, \boldsymbol{s_v}), U, V) \\ &= (\boldsymbol{s_u}, \boldsymbol{s_v} \to (\boldsymbol{s_u}, \boldsymbol{s_v}), U, V). \end{aligned}$$

Consider a convolution $F * G$ of vectors $F \in \mathbb{R}^2$ and $G \in \mathbb{R}^3$. Then

$$(F * G)_x = \sum_{y \in [2]} F_y G_{x+2-y}$$

for $x \in [2]$, and

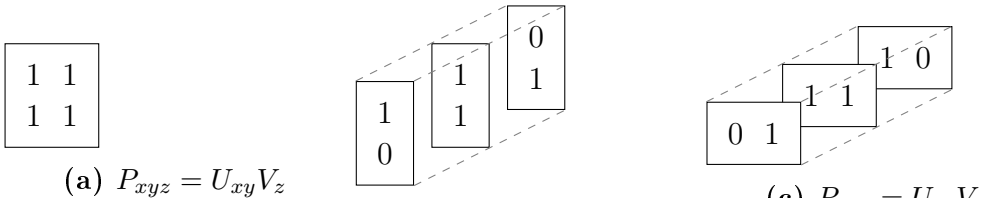
$$P_{(x,y,z)} = \begin{cases} 1 & \text{if } z = x + 2 - y \\ 0 & \text{else} \end{cases}$$

for $x \in [2], y \in [2], z \in [3]$. This is illustrated in Figure 5.1.

Now, the only two possibilities for m are $m = 1$ and $m = 2$, because $n = 1$ and therefore $1 \leq m < 3$. This means, U and V are a matrix and a vector. W.l.o.g. we assume that U is a matrix and V is a vector. Then for U to be a matrix and a factor of P, there are only three possibilities:

$$\begin{aligned} P_{xyz} &= U_{xy} V_z, \\ P_{xyz} &= U_{xz} V_y, \\ P_{xyz} &= U_{yz} V_x. \end{aligned}$$

In the first case, it has to hold that $U_{xy} = 1$ where $P_{xyz} = 1$ for any $z \in [3]$, because otherwise, no design tensor V could produce the entry $P_{xyz} = 1$. For the other two cases, the analogous fact holds: all the entries, where missing index exists such that $P_{xyz} = 1$, have to hold one as well. And unless V is full of zeros (which it cannot be), the converse is true as well, meaning if all entries of the missing index hold $P_{xyz} = 0$, then the entry in U also has to hold zero. Therefore there is only possibility for U for each of the cases, which



(a) $P_{xyz} = U_{xy}V_z$ **(b)** $P_{xyz} = U_{xz}V_y$ **(c)** $P_{xyz} = U_{yz}V_x$

Figure 5.2: Possiblities for U in the factorization of P

are illustrated in Figure 5.2:

$$P_{xyz} = U_{xy}V_z \qquad U = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$P_{xyz} = U_{xz}V_y \qquad U = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$P_{xyz} = U_{yz}V_x \qquad U = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Therefore, the multiplication with V has to spread out the ones across different values of the missing index, which is not possible, because depending on the entries of V on the missing index:

- if V is one at a value of the missing index, then P is a copy of U at this value of the missing index, and
- if V is zero at a value of the missing index, then P is zero at this value of the missing index.

Therefore, there is no factorization of P when $F \in \mathbb{R}^2$ and $G \in \mathbb{R}^3$, which means P can not generally be expressed as an outer product of two lower-order tensors. □

5.5 Max-Pooling

Let T be an n-th order tensor where all axes have size m. Let $p \in \mathbb{N}$ be the *pool size* such that $m = k \cdot p$ for $k \in \mathbb{N}$. Then the max-pooling of T is defined in the following way:

$$M(T)_x := \max_{y \in [p]^n} T_{p(x-1)+y}$$

for $x \in [k]^n$, where $x - 1$ indicates the element-wise substraction $(x-1)_i = x_i - 1$ for $i \in [n]$. Let

$$T'_{(x,y)} = T_{p(x-1)+y}$$

for $x \in [k]^n, y \in [p]^n$. Then

$$M(T)_x = \max_{y \in [p]^n} T'_{(x,y)}$$

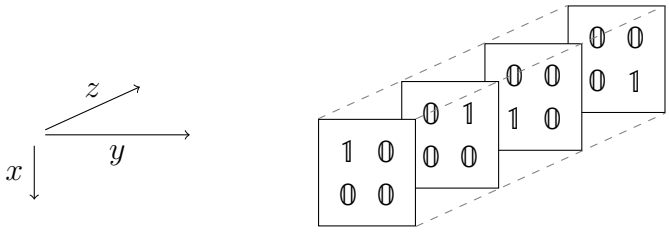


Figure 5.3: P for $F \in \mathbb{R}^2$ and $G \in \mathbb{R}^3$

for $x \in [k]^n$. Let

$$P_{(x,y,z)} := \begin{cases} 0 & \text{if } z = p(x-1)+y \\ -\infty & \text{else} \end{cases}$$

for $x \in [k]^n, y \in [p]^n, z \in [m]^n$. Then

$$T'_{(x,y)} = \max_{z \in [m]^n} P_{(x,y,z)} + T_z$$

for $x \in [k]^n, y \in [p]^n$. Therefore, Max-Pooling can be expressed as an Einsum expression:

$$M(T) = ((\boldsymbol{s_x}, \boldsymbol{s_y}, \boldsymbol{s_z}), \boldsymbol{s_z} \to \boldsymbol{s_x}, P, T)_{R_{(\max,+)}}$$

where $\boldsymbol{s_x}, \boldsymbol{s_y}, \boldsymbol{s_z} \in S^n$ use distinct symbols and $R_{(\max,+)}$ denotes the tropical semiring $(\mathbb{R} \cup \{-\infty\}, \max, +)$.

As in Section 5.4, the efficiency of the computation could benefit from decomposition of the design tensor P into an outer product of lower-order design tensors. But yet again, this is not possible.

Proof. Because $-\infty$ and 0 are the additive neutral and multiplicative neutral element of the used semiring respectively, the argument in Section 5.4, that the factors have to be scaled design tensors, holds here as well.

Therefore, w.l.o.g. we assume that U and V are design tensors of order one or higher. Consider max-pooling where $n = 1$, $m = 4$, $p = 2$, and $k = 2$. Then

$$P_{(x,y,z)} := \begin{cases} 0 & \text{if } z = 2(x-1)+y \\ -\infty & \text{else} \end{cases}$$

for $x \in [2], y \in [2], z \in [4]$. This is illustrated in Figure 5.3, where $\mathbb{0}$ and $\mathbb{1}$ were used to indicate the additive and multiplicative neutral element of the tropical semiring, $-\infty$ and 0.

The rest of the proof is analogous to the proof in Section 5.4, where the key argument is, that the multiplicative neutral element cannot be distributed with an outer product in such a way, that there are no copied slices in P. □

6 Practical Results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7 Discussion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

8 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bibliography

[1] Srinivas Aji. Graphical models and iterative decoding. 01 2000.

[2] Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. 2018. doi: 10.48550/ARXIV.1805.07091.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

List of Figures

List of Tables

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Seitens des Verfassers bestehen keine Einwände die vorliegende Bachelorarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Jena, 13.07.2023