Factoring Tensor Product Expressions: How Hard Could it Be?

$$a \otimes b + a \otimes c \to a \otimes (b + c)$$

Jack Beda : jack@beda.ca

Background

When computing the probabilities for certain outcomes of particle collider experiments to occur, a particular class of functions keeps appearing. These functions, the *multiple polylogarithms*, satisfy a deep and mysterious algebraic structure. One tool in the examination of this structure is the *coproduct*, which takes a complicated multiple polylogarithm expression and breaks it into simpler parts. The price we pay for this decomposition is that often the decomposition contains many terms, sometimes more than 20 000. It is essential to make this more manageable by factoring the expression to a minimum number.

We can abstractify the problem away from the context of particle physics to the purely mathematical realm. Instead of considering the vector space of multiple polylogarithmic expressions, we consider an arbitrary vector space. Instead of the coproduct, we consider an arbitrary tensor product expression. We would like an algorithm that can minimally factor (i.e. factor to a minimum number of terms) expressions like:

$$x_0 = a_1 \otimes b_1 + a_1 \otimes b_3 + a_2 \otimes b_2 + a_2 \otimes b_3 \longrightarrow a_1 \otimes (b_1 + b_3) + a_2 \otimes (b_2 + b_3) \quad (1)$$

$$x_1 = a_1 \otimes b_1 + a_1 \otimes b_2 + a_2 \otimes b_1 + a_2 \otimes b_2 \longrightarrow (a_1 + a_2) \otimes (b_1 + b_2) \quad (2)$$

The Problem

Let V be a vector space. Let $\{v_i\}, \{u_j\} \subset V$ for $i = 1, 2, \ldots, n_1$, $j = 1, 2, \ldots, n_2$ be linearly independent subsets. Given an unfactored tensor product expression:

$$x = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} c_{ij} (v_i \otimes u_j) = c_{11} v_1 \otimes u_1 + c_{12} v_1 \otimes u_2 + \cdots + c_n v_n \otimes u_n. \quad (3)$$

What is the minimum $r \in \mathbb{Z}$, such that there exists $d_{ij}^{(1)}$ and $d_{ij}^{(2)}$ such that we may factor x as:

$$x = \sum_{l=1}^{r} \left(\sum_{i=1}^{n_1} d_{li}^{(1)} v_i \right) \otimes \left(\sum_{j=1}^{n_2} d_{lj}^{(2)} u_j \right). \quad (4)$$

Naive Algorithm

Let's suppose the expression we need to factor is:

$$x_0 = a_1 \otimes b_1 + a_2 \otimes b_2 + a_1 \otimes b_3 + a_2 \otimes b_3. \quad (5)$$

A naive algorithm might try grouping left terms $(a \otimes c + b \otimes c \to (a + b) \otimes c)$, then right terms $(a \otimes b + a \otimes c \to a \otimes (b + c))$, until no further terms match. Applying such an algorithm starting starting from the left would give:

$$x_0 = a_1 \otimes b_1 + a_2 \otimes b_2 + (a_1 + a_2) \otimes b_3 \quad \text{(3 terms)} \quad (6)$$

Or from the right:

$$x_0 = a_1 \otimes (b_1 + b_3) + a_2 \otimes (b_2 + b_3) \quad \text{(2 terms)} \quad (7)$$

It turns out the factorization with 2 terms is minimal, which isn't too hard to see from this example (give it a try!), but we already see the limitations of this algorithm: *it can get stuck*. The factorization in Eq. 6 contains no terms that can be grouped, and yet is not minimal. This algorithm is *greedy*: it always takes the locally optimal decision, but we have no guarantee that the final factorization it provides is minimal.

Factoring Algorithm for one Tensor Product

A better algorithm can be acquired by setting the equations for the factored (Eq. 3) and unfactored (Eq. 4) forms an arbitrary expression x equal and rearranging:

$$x = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} c_{ij} (v_i \otimes u_j) = \sum_{l=1}^{r} \left(\sum_{i=1}^{n_1} d_{li}^{(1)} v_i \right) \otimes \left(\sum_{j=1}^{n_2} d_{lj}^{(2)} u_j \right) \quad (8)$$

$$= \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left(\sum_{l=1}^{r} d_{li}^{(1)} d_{lj}^{(2)} \right) (v_i \otimes u_j) \quad \text{(Expanding)} \quad (9)$$

$$\implies 0 = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left(c_{ij} - \sum_{l=1}^{r} d_{li}^{(1)} d_{lj}^{(2)} \right) (v_i \otimes u_j) \quad (10)$$

$$\implies c_{ij} = \sum_{l=1}^{r} d_{li}^{(1)} d_{lj}^{(2)}. \quad (11)$$

Identifying the matrices C, $D^{(1)}$, and $D^{(2)}$ with c_{ij}, $d_{ij}^{(1)}$, and $d_{ij}^{(2)}$ this is saying that a factorization of x in r terms is possible if and only if the coefficient matrix C can be written as a product of two other matrices of appropriate dimensions:

$$\underset{n_1 \times n_2}{C} = \left(\underset{n_1 \times r}{D^{(1)T}} \right) \left(\underset{r \times n_2}{D^{(2)}} \right). \quad (12)$$

Where "$\underset{n \times m}{A}$" is shorthand for "the $n \times m$ matrix A". A result from linear algebra is that $r = \mathrm{Rank}(C)$ is the smallest integer r such that Eq. 12 holds. **Thus we have shown that the minimum number of terms in a factorization of x is simply the rank of the coefficient matrix C.** Actually finding the factorization (i.e. finding $D^{(1)}$ and $D^{(2)}$), can be done in various ways. The computationally fastest is to let R be the reduced row echelon form of C. Then let $D^{(1)T}$ be matrix formed of all pivot columns of C and $D^{(2)}$ be the matrix formed of all rows of R that are not entirely zero.

Examples

Let's return to the example where we claimed x_0 had a minimum factorization with 2 terms:

$$x_0 = a_1 \otimes b_1 + a_2 \otimes b_2 + a_1 \otimes b_3 + a_2 \otimes b_3 \longrightarrow a_1 \otimes (b_1 + b_3) + a_2 \otimes (b_2 + b_3). \quad (13)$$

In this case the two independent subsets of our vector space are $\{a_1, a_2\}$ and $\{b_1, b_2, b_3\}$. The unfactored expression has coefficient matrix:

$$c_{ij} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}_{ij}. \quad (14)$$

Indeed you can check that:

$$x_0 = c_{11}\, a_1 \otimes b_1 + c_{21}\, a_2 \otimes b_1 + c_{12}\, a_1 \otimes b_2 + \cdots \quad (15)$$

Importantly, we see that $\mathrm{Rank}(C) = 2$, and so 2 terms is indeed the minimal number of terms in a factorization of x_1. As another example, the coefficient matrix of the unfactored expression form Eq. 2 for x_1 is:

$$c_{ij} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}_{ij}. \quad (16)$$

Which has rank 1, as we expect, because it can be factored into a single term. These are simple examples, but the power of this method is much greater, for example, the coproduct of the function containing more than 20 000 terms can be factored into as few as 21 terms.

Generalization to More Tensor Products

We can now minimally factor expressions with one tensor product symbol. How about more? That is, for some coefficient *tensor* of order m: $c_{i_1 i_2 \cdots i_m}$, what is the minimal factorization of:

$$y = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_m=1}^{n_m} c_{i_1 i_2 \cdots i_m} \left(v_{i_1}^{(1)} \otimes v_{i_2}^{(2)} \otimes \cdots \otimes v_{i_m}^{(m)} \right). \quad (17)$$

That is, what is the minimum r such that:

$$y = \sum_{l=1}^{r} \left(\sum_{i_1=1}^{n_1} d_{li_1}^{(1)} v_{i_1}^{(1)} \right) \otimes \left(\sum_{i_2=1}^{n_2} d_{li_2}^{(2)} v_{i_2}^{(2)} \right) \otimes \cdots \otimes \left(\sum_{i_m=1}^{n_m} d_{li_m}^{(m)} v_{i_m}^{(m)} \right). \quad (18)$$

As before, expanding Eq. 18 and comparing the with Eq. 17 yields the relationship:

$$c_{i_1 i_2 i_3 \cdots i_m} = \sum_{l=1}^{r} d_{li_1}^{(1)} d_{li_2}^{(2)} \cdots d_{li_m}^{(m)}. \quad (19)$$

It turns out the minimum r is again given by $\mathrm{Rank}(C)$, where here we mean *tensor rank*. Note that in the field of tensor decomposition, we use the term tensor *order* to refer to the number of indices on a tensor, where the rank of a tensor is totally different, and much harder to compute.

Properties of the Tensor Decomposition

Unfortunately, it turns out that computation of tensor rank, let alone tensor decompositions, for tensors of order greater than 2 is NP-hard. That is to say, the Clay Math Institute will give anyone with a polynomial time algorithm a million dollars for having showing $P = NP$.

Another surprising property is that tensor rank of real tensors (of order greater than 2) can differ over $\mathbb{R}$ and over $\mathbb{C}$. For example, the order 2 tensor in Fig. 1 has rank 3 over $\mathbb{R}$ but rank 2 over $\mathbb{C}$. In the case of factoring, this means that the real expression:

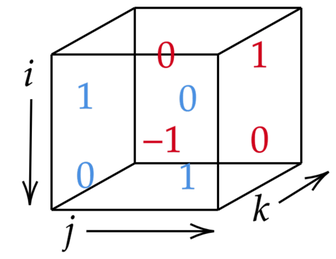

Figure 1: *Coefficient tensor of* y_0.

$$y_0 = u_1 \otimes v_1 \otimes w_1 + u_1 \otimes v_2 \otimes w_2 - u_2 \otimes v_1 \otimes w_2 + u_2 \otimes v_2 \otimes w_1. \quad (20)$$

Can be minimally factored to three terms over $\mathbb{R}$:

$$y_0 = u_1 \otimes v_1 \otimes (w_1 - w_2) + u_2 \otimes v_2 \otimes (w_1 + w_2) + (u_1 - u_2) \otimes (v_1 + v_2) \otimes w_2. \quad (21)$$

But minimally factored to two terms over $\mathbb{C}$:

$$y_0 = \frac{1}{2}((u_1 - iu_2) \otimes (v_1 + iv_2) \otimes (w_1 - iw_2) + (u_1 + iu_2) \otimes (v_1 - iv_2) \otimes (w_1 + iw_2)). \quad (22)$$

.

Conclusion

We have presented an efficient algorithm for minimally factoring tensor product expressions. In particular, this can be used to find a basis of functions appearing in the coproduct. However, we have seen that (assuming $P \neq NP$) there is no polynomial time algorithm for minimally factoring expressions involving more than one tensor product. In particular, this means that finding a basis of functions of the *symbol* is much more challenging. That said, there do exist *exponential* time algorithms for minimal factorizations. This is how Eq. 22 was acquired, but these are far too slow for expressions with as few as 9 terms. We note here that in the case of the multiple polylogarithms, since multiple polylogarithms satisfy numerous identities, i.e. they are not linearly independent from one another, finding a basis involves more than just a minimal factorization.