week2 sumarry

Mohamed Hamlil

June 2024

1 Introduction

First we will present the math tools and how is being used in our algorithms then we will see the version of BDEZ algorithm and other generic algorithm using tensors and how we have extended and optimized them

2 Model Approach

We will see how to use tensors for bilinear maps and that studying there behavior is equivalent to studying there associated tensor(that will see in this section) then how to use it on bilinear rank problem.

2.1 Multilinear Maps and Tensors

A tensor is an algebraic object that describes a multilinear relationship between sets of algebraic objects related to a vector space.
It's a generalized notion of matrices that we have used to modelize bilinear maps. For example:

Consider a bilinear map $f : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ given by:

$$f((x_1, x_2), (y_1, y_2)) = x_1y_1 + 2x_1y_2 + 3x_2y_1 + 4x_2y_2$$

Matrix Representation

$$f((x_1, x_2), (y_1, y_2)) = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

Tensor (2 Covariant) Representation

$$f((x_1, x_2), (y_1, y_2)) = \left[[[1, 2], [3, 4]]\right] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Multilinear Maps

A multilinear map is a function that is linear in each of its arguments separately. Specifically, an r-multilinear map f is a map of the form:

$$f : V_1 \times V_2 \times \cdots \times V_r \to W$$

where $V_1, V_2, \ldots, V_r$ are vector spaces over a field K, and W is another vector space over the same field. The map f is called multilinear if it satisfies the following property for each i (and similar properties for other arguments):

$$f(v_1, \ldots, av_i + bw_i, \ldots, v_r) = af(v_1, \ldots, v_i, \ldots, v_r) + bf(v_1, \ldots, w_i, \ldots, v_r)$$

for all $v_i, w_i \in V_i$ and scalars $a, b \in K$.

Tensors

A tensor can be viewed as a generalization of vectors and matrices to higher dimensions. Formally, a tensor is an element of a tensor product of vector spaces. Specifically, an (r, s)-tensor T is an element of the space:

$$T \in V_1 \otimes V_2 \otimes \cdots \otimes V_r \otimes V_1^* \otimes V_2^* \otimes \cdots \otimes V_s^*$$

2.2 Relationship

Representation of Multilinear Maps as Tensors Every multilinear map can be represented by a corresponding tensor. Given an r-multilinear map:

$$f : V_1 \times V_2 \times \cdots \times V_r \to W$$

we can associate a tensor T_f such that:

$$T_f \in V_1^* \otimes V_2^* \otimes \cdots \otimes V_r^* \otimes W$$

This tensor T_f encodes the information of the multilinear map in its components.

Components of Tensors If $\{e_i^{(k)}\}$ are bases for V_k and $\{e_{(k)}^i\}$ are the dual bases for V_k^*, then the components of the tensor T associated with a multilinear map f can be written as:

$$T^j_{i_1 i_2 \cdots i_r} = f(e^{(1)}_{i_1}, e^{(2)}_{i_2}, \ldots, e^{(r)}_{i_r})$$

Here, the indices $i_1, i_2, \ldots, i_r$ correspond to the bases of the vector spaces, and j corresponds to the basis of the target space W.

Evaluation of Tensors Conversely, a tensor $T \in V_1^* \otimes V_2^* \otimes \cdots \otimes V_r^* \otimes W$ defines a multilinear map:

$$f_T(v_1, v_2, \ldots, v_r) = T(v_1 \otimes v_2 \otimes \cdots \otimes v_r)$$

Here, the tensor T is evaluated on the tensor product of vectors $v_1, v_2, \ldots, v_r$ from the respective vector spaces.

Examples

Example 1: Bilinear Map Consider a bilinear map $f : V \times V \to W$. It can be represented as a tensor $T \in V^* \otimes V^* \otimes W$. If $\{e_i\}$ is a basis for V and $\{e^i\}$ is the dual basis, then the components T_{ij}^k of the tensor are given by:

$$T_{ij}^k = f(e_i, e_j)$$

The bilinear map f can be recovered from the tensor T as:

$$f(v, w) = \sum_{i,j,k} T_{ij}^k v^i w^j e_k$$

Example 2: Trilinear Map For a trilinear map $g : U \times V \times W \to X$, it can be represented as a tensor $S \in U^* \otimes V^* \otimes W^* \otimes X$. If $\{u_i\}$, $\{v_j\}$, $\{w_k\}$ are bases for U, V, W respectively, and $\{x_l\}$ is a basis for X, then:

$$S_{ijk}^l = g(u_i, v_j, w_k)$$

The trilinear map g can be recovered from the tensor S as:

$$g(u, v, w) = \sum_{i,j,k,l} S_{ijk}^l u^i v^j w^k x_l$$

Conclusion

The relationship between multilinear maps and tensors is that every multilinear map can be uniquely represented by a tensor whose components encode the map's action on basis elements. Conversely, a tensor defines a multilinear map through the evaluation of the tensor on the tensor product of input vectors. This duality makes tensors a powerful tool for studying and working with multilinear maps.

2.3 Minimizing the number of multiplication in a bilinear map

we will take polynomial multiplication as an example to explain our approach and how we can use some tools(Vandermonde matrix) to minimze the computation

2.3.1 reformultion of the problem using tensors

Let $A(x)$ and $B(x)$ be polynomials with coefficients from a field K of degrees n and m respectively.
Multiplying the two polynomials can be represented as a bilinear map:

$$K^n \times K^m \to K^l$$

$$((a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{m-1})) \mapsto (c_0, \ldots, c_{l-1})$$

The bilinear map can be represented as a T_2^1 tensor with 0 or 1 as coefficients. For example, let $A(x) = a_0 + a_1x$ and $B(x) = b_0 + b_1x + b_2x^2$. It can be represented as:

$$\begin{bmatrix} [1,0,0] & [0,0,0] \\ [0,1,0] & [1,0,0] \\ [0,0,1] & [0,1,0] \\ [0,0,0] & [0,0,1] \end{bmatrix}$$

This tensor we will call the associated tensor (similar to how linear maps have associated matrices), where:

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} [[1,0,0],[0,0,0]] \\ [[0,1,0],[1,0,0]] \\ [[0,0,1],[0,1,0]] \\ [[0,0,0],[0,0,1]] \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

we can see that in each row represent the multiplications used to compute each c term

2.3.2 identifiifying the problem

Now we will try to find a decomposition where $T = \phi \cdot T'$. We will continue with the same example. The T tensor can be decomposed as:

$$\begin{bmatrix} 1,0,0,0,0 \\ -1,0,-1,0,1 \\ 0,1,1,0,0 \\ 0,0,0,1,0 \end{bmatrix} \cdot \begin{bmatrix} [[1,0,0],[0,0,0]] \\ [[0,0,1],[0,0,0]] \\ [[0,0,0],[0,1,0]] \\ [[0,0,0],[0,0,1]] \\ [[1,1,0],[1,1,0]] \end{bmatrix}$$

Let $A(x)$ and $B(x)$ be polynomials with coefficients from a field K of degrees n and m respectively.
Multiplying the two polynomials can be represented as a bilinear map:

$$K^n \times K^m \to K^l$$

$$((a_0, \ldots, a_{n-1}), (b_0, \ldots, b_{m-1})) \mapsto (c_0, \ldots, c_{l-1})$$

The bilinear map can be represented as a T_2^1 tensor with 0 or 1 as coefficients. For example, let $A(x) = a_0 + a_1x$ and $B(x) = b_0 + b_1x + b_2x^2$. It can be represented as:

$$\begin{bmatrix} [[1,0,0],[0,0,0]] \\ [[0,1,0],[1,0,0]] \\ [[0,0,1],[0,1,0]] \\ [[0,0,0],[0,0,1]] \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

This setup is inspired by the Karatsuba trick:

$$C(x) = (a_1 \cdot x + a_0) \times (b_2 \cdot x^2 + b_1 \cdot x + b_0)$$

$$= a_1b_2 \cdot x^3 + (a_1b_1 + a_0b_2) \cdot x^2 + (a_0b_1 + a_1b_0) \cdot x + a_0b_0$$

Five products are required instead of six. Using Karatsuba's trick:

$$C(x) = a_1b_2 \cdot x^3 + (a_1b_1 + a_0b_2) \cdot x^2 + ((a_0 + a_1)(b_0 + b_1) - a_1b_1 - a_0b_0) \cdot x + a_0b_0$$

Compute the products:

$$\begin{aligned} g_0 &= a_0 \cdot b_0, \\ g_1 &= a_0 \cdot b_2, \\ g_2 &= a_1 \cdot b_1, \\ g_3 &= a_1 \cdot b_2, \\ g_4 &= (a_0 + a_1) \cdot (b_0 + b_1). \end{aligned}$$

Reconstruct the result:

$$C(x) = g_3 \cdot x^3 + (g_1 + g_2) \cdot x^2 + (g_4 - g_2 - g_0) \cdot x + g_0$$

The T' tensor requires only five multiplications instead of six.
The key observation is that in the last row of T', the first and second components are the same, which means they can be taken as a common factor, reducing the number of multiplications to one. From this, we conclude that the number of multiplications in a given row is the number of linearly independent components.
For an all-zero vector, no computation is needed by (definition is linearly dependent). Linearly dependent vectors can be expressed by other vectors, thus reducing the number of multiplications in a row.

In simplier terms, the number of multiplications in a given row is the rank of the matrix made of the components (covectors) of that row. We will call that matrix the row matrix and denote for a given tensor A that A_i is the ith row matrix.

Therefore, the total number of multiplications is equal to the sum of the ranks of the row matrices. Our objective is to find a decomposition $T = \phi \cdot T'$ where:

$$\arg\min \left(\sum \operatorname{rank}(T'_i) \right)$$

Conclusion
We need a generating family T' where $\arg\min \left(\sum \operatorname{rank}(T'_i) \right)$ and it generates the same vector space as T.
So Minimizing the number of multiplication in a bilinear map T is equivalent to find a decomposition T' such that it spans T and:

$$\arg\min \left(\sum \operatorname{rank}(T'_i) \right)$$

2.4 Finding Optimal Formulae for polynomial multiplication Using Vandermonde matrix Over a Finite field

we continue our road of adapting and generalzing our Tensor tools for bilinear map(the case of polynomial multiplication)
Let $A(x)$ and $B(x)$ be polynomials with coefficients from a field K of degrees n and m respectively.
Corollary: Let $C(x)$ be the product of $A(x)$ and $B(x)$, then the degree of $C(x)$ is $m+n$, which implies that we will need at least $m+n$ multiplications.
So, $m+n$ is a lower bound on the number of multiplications required to multiply $A(x)$ and $B(x)$.

Interpolation Multiplication

Interpolation multiplication involves generating $m+n$ points from $A(x)$ and $B(x)$, then multiplying their images in $m+n$ operations, followed by reverse interpolation to get the coefficients of $C(x)$.

Tensor Representation

The interpolation and reverse interpolation can be represented by a Vandermonde matrix and its inverse respectively.
Let $C'(x)$ be the product of the images of $A(x)$ and $B(x)$. Then the coefficients of $C(x)$ can be written in terms of its images by multiplying the images with the inverse of the Vandermonde matrix:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m+n} \end{bmatrix} = \begin{bmatrix} \text{Inverse Vandermonde matrix} \end{bmatrix} \begin{bmatrix} c'_0 \\ c'_1 \\ \vdots \\ c'_{m+n} \end{bmatrix}$$

The product between the images of $A(x)$ and $B(x)$ can be represented as a kind of identity T_2^1 tensor:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m+n} \end{bmatrix} = \begin{bmatrix} \text{Inverse Vandermonde matrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} A'_0 \\ A'_1 \\ \vdots \\ A'_{m+n} \end{bmatrix} \begin{bmatrix} B'_0 \\ B'_1 \\ \vdots \\ B'_{m+n} \end{bmatrix}$$

which are both interpolations of $A(x)$ and $B(x)$:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m+n} \end{bmatrix} = \begin{bmatrix} \text{Inverse Vandermonde matrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} \text{Vandermonde matrix} \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{m+n} \end{bmatrix} \begin{bmatrix} \text{Vandermor} \end{bmatrix}$$

So by what we have seen, $T = \phi \cdot T'$ becomes finding $T = \phi^{-1} \cdot T' \cdot \phi' \cdot \phi''$.
Proposition: This tensor decomposition is one of the methods to get the optimal solution.
Proof: This tensor representation, which will decompose into a matrix multiplied with a tensor and then 2 matrices $T = \phi^{-1} \cdot T' \cdot \phi \cdot \phi$, requires only $m + n$ multiplications, which is the minimum we can get, as we have seen at the beginning.
interpolation/vandermonde trick over prime field P for polynomial multiplication

Interpolation works well for polynomial multiplication, but finite fields have a limitation: they do not provide enough points to generate a sufficient number of images. To find the optimal multiplication formula for polynomial multiplication, we need a combination of interpolation and generic algorithms.

In a prime field P, the maximum degree for which interpolation multiplication can be performed is $P-1$so the set of products of images using the elements of the primes fields as points is in our final solution and can be used as base for our generic algorithm.

3 Optimizing Generic Algorithms for the Bilinear Rank Problem Using Tensors

In this section, we will discuss the algorithms mentioned in Finding Optimal Formulae for Bilinear Maps and BDEZ, analyze their problems, and present our algorithm along with comparisons and improvements.

The BDEZ downsides start with the fact that these algorithms work with vector spaces instead of bases, which can lead to several issues:

- A multitude of solution vector spaces, even if they are spanned by the same base.
- Representation of vector spaces is more memory-intensive.
- Inefficiency of some algorithms (e.g., HasRankOneBasis(V)) as mentioned in BDEZ, and recursive search of unnecessary possibilities, for example, Algorithm 1 in Finding Optimal Formulae for Bilinear Maps:

```
dim W = k and rk(W  G) = k then
4. S ← S  {W}
5. else if dim W < k then
6. for each g  G \ W do
7. expand subspace(W  Span(g))
```

where:

- `dim W` represents the number of vectors in our solution.
- `rk(W)` represents the rank of W.

3.1 Improvement and Adaptation

We will start by discussing how to generate G (the set of all bilinear rank-1 tensors). The algorithm creates all possible bilinear rank-1 tensors in a finite field by building on the previously generated bilinear rank-1 tensors, maintaining the condition that they have rank 1.

3.2 The Algorithm

First Version:

```
def expand_subspace(W, G, T):
    if Number_of_multiplications(W) == k and Span_it(W, T):
        return W
    if Number_of_multiplications(W) < k:
        for g in range(len(G)):
            if is_linearly_independent(W, G[g]) and
               (expand_subspace(W + [G[g]], G[g + 1:], T) is not None):
                return expand_subspace(W + [G[g]], G[g + 1:], T)
    return None
```

The algorithm goes through the possible bases that generate T while the number of multiplications equals k. The improvements we have adapted and added facilitate the `HasRankOneBasis(V)` function because G contains only rank-1 tensors. We combine the conditions `dim W` and `rk W` into a more general one: linear independence, reducing the number of iterations and memory usage (vector spaces consume more memory).

3.3 Uniqueness of the Solution Found by Our Algorithm

Firstly, finding the formulas for the inputs in a classical way (e.g., input $[a_0, a_1], [b_0, b_1, b_2]$ outputs $[a_0b_0, (a_0b_1 + a_1b_0), (a_1b_1 + a_0b_2), a_1b_2]$) of the bilinear map. Since our solution is in tensor form, the formulas are all concatenated. To find all possible formulas, we need to perform all possible tensor decompositions of each row matrix. Secondly, our solution T' generates T, and we know that applying automorphisms keeps the rank (number of multiplications) the same. Thus, all solutions are related by applying an automorphism, and finding all possible formulas is equivalent to exploring most automorphism versions and tensor decompositions of each row.

3.4 Binary Search Implementation

Instead of iterating over all possibilities to find the minimal multiplication, we use a binary search. Since we need to find only one solution, this is more appropriate.

3.5 Bottom-Up Approach

The algorithm builds the solution starting the matrices of a lower rank to span those with higher rank in T. In this version, we sort the rows of T according to their rank, then build up a solution that spans the rows of the lowest rank. We use the solution to build it up such that it spans the rows of the next rank. Continuously, we take the previous solution as a base and complete it until it spans the next row matrices of the next rank.

3.6 Beta Version

improvement using automorphisms:

```
def rp_expand_subspace(W, G, T):
    if Number_of_multiplications(W) == k and Span_it(W, T):
        return W
    if Number_of_multiplications(W) < k:
        for g in range(len(G)):
            if is_linearly_independent(W, G[g]) and
               (not rp_sort(W, G[g])) and
               (rp_expand_subspace(W + [G[g]], G[g + 1:], T) is not None):
                return rp_expand_subspace(W + [G[g]], G[g + 1:], T)
    return None
```

We tried applying the concept of `RP_automorphism` such that:

$$T$$

is a bilinear map and we will take a special kind of automorphism. This automorphism, instead of having numbers, has automorphisms. For example:

$$\begin{bmatrix} [\text{random rank-1 tensor}] \\ [\text{random rank-1 tensor}] \\ \vdots \\ [\text{random rank-1 tensor}] \end{bmatrix} \begin{bmatrix} \text{automorphism matrix 1} \end{bmatrix} \otimes \begin{bmatrix} \text{automorphism matrix 2} \end{bmatrix}$$

This automorphism preserves the rank and vector space of the tensor. If automorphism matrix 2 is the identity matrix, applying the special automorphism would be equivalent to performing a linear combination.
In the first algorithm, we observed that there is no need to go through the rank-1 tensors that are linearly dependent. Therefore, if we fix automorphism matrix 1 as the identity matrix, there is no need to go through rank-1 tensors that are images of other rank-1 tensors by an automorphism.
The most general and optimal way is by solving a system of linear equations instead of separately verifying linear independence and automorphisms.

Minimize the number of multiplications of a given tensor while preserving its dimension:

The tensor represents a basis of bilinear maps (represented by matrices). The question that arises is whether this basis has the least number of multiplications while keeping the number of its elements constant.

We present an algorithm that, given this basis, will return the minimum number of multiplications required while keeping the number of elements constant.

We start by choosing the bilinear application j in the basis with the highest rank and trying all the possible linear combinations using all the elements of this basis under the constraint that the scalar multiplied with bilinear application j must not be zero. After this, in the set of these linear combinations, we will choose the bilinear application j (matrix) such that its rank is the minimum and less than the rank of the bilinear application j and exchange between the bilinear application j and the minimum.

Thus, we move to the second highest rank, doing this on all the bilinear applications in T (the order won't affect the result because they are linear combinations and preserve symmetries). We will end up with a basis T' that requires the least number of multiplications with the same number of elements in T (because if we suppose the existence of a basis with fewer multiplications, it means that there is a linear combination that takes us to it, but as we have seen all the possible linear combinations and chosen the minimum, it is absurd to find one with less).

This algorithm will reduce the depth of our recursion in the recursion algorithm and the number of branches in our recursion tree such that:

Given a bilinear application of rank 1 v and a basis T, v_1 won't be in our optimal solution if:

number of multiplications of base_reducer(T+v) > number of multiplications of base_reducer(T)

(The algorithm that reduces the number of multiplications of a basis while preserving the number of elements will be called *base_reducer*.)

Proof:

Let's take 2 rank-1 bilinear applications called v_1 from G (the set of all possible bilinear applications of rank 1) such that:

number of multiplications of base_reducer(T+v_1) ≥ number of multiplications of base_reducer(T)

number of multiplications of base_reducer(T+v_2) ≥ number of multiplications of base_reducer(T)

number of multiplications of base_reducer(T+v_1+v_2) ≤ number of multiplications of base_reducer(T)

The span(v_1, v_2) will have a rank 2 bilinear application or rank 1 bilinear application.

If it is rank 2, it implies the matrix rank 1 decomposition of one of the bilinear maps in T and they can't be in many decompositions of one of the bilinear maps in T because if so:

number of multiplications of base_reducer(T+v_1) < number of multiplications of base_reducer(T)

or

number of multiplications of base_reducer(T+v_2) < number of multiplications of base_reducer(T)

If it is rank 1, then we will choose this bilinear application from instead of v_1 or v_2.

So this proves that using *base_reducer* will reduce the time complexity of the first generic algorithms.

NP-completeness

Instead of using elements from G, we will use the vector space that is linearly independent from the set of rank 1 bilinear applications of a basis T.

So we will have 2 validators to validate our solution (minimal number of multiplications of a given basis). The first validates if a combination of a rank 1 linearly independent bilinear map v satisfies the condition:

number of multiplications of base_reducer(T+v) < number of multiplications of base_reducer(T)

Then we must find a suite of rank 1 linearly independent bilinear maps v_i such that:

The last validator will return true if:

$$\text{number of multiplications of base\_reducer}(T + v_1 + v_2 + \ldots + v_n) = k$$

In other words, finding a linear combination i of the set of rank 1 bilinear applications that is linearly independent from the set of rank 1 bilinear applications of a basis T_i such that it satisfies the condition:

number of multiplications of base_reducer(T_i+v_i) < number of multiplications of base_reducer(T_i)

Then recursively until it validates:

$$\text{number of multiplications of base\_reducer}(T + v_1 + v_2 + \ldots + v_n) = k$$

Recursion won't affect the time complexity and the choice of v_i and the validators are NP-complete and generating a bilinear application of rank 1 that is linearly independent from the set of rank 1 bilinear applications of the basis T_i is in polynomial time, which implies the problem is NP-complete.

4 Conclusion

our algorithm using tensors let us go throw bases instead of vector spaces and reduce the number of solution into one that can generate all of them while keeping a wide range of improvement