On Tensor Decompositions

Hua-Lin Huang
Huaqiao University

TSIMF 清华三亚国际数学论坛

丘成桐数学科学中心 YAU MATHEMATICAL SCIENCES CENTER

霍普夫代数与张量范畴国际研讨会

International Conference on Hopf Algebras and Tensor Categories

On Tensor Decompositions

Hua-Lin Huang
Huaqiao University

What is the talk about?

What is the talk about?

* At the categorical level, or the macro level, one usually decomposes objects into sums of simpler objects. Typical examples appear in the theories of representation rings, Brauer groups, branching laws, etc.

What is the talk about?

- At the categorical level, or the macro level, one usually decomposes objects into sums of simpler objects. Typical examples appear in the theories of representation rings, Brauer groups, branching laws, etc.

- In many problems, one needs to consider explicit elements in a given object. For example, additive and multiplicative decompositions of polynomials are ubiquitous. This talk is about decompositions of tensor elements at the micro level.

Tensor Element Decompositions

Tensor Element Decompositions

* **Main Problem**: Given an element in a tensor space, or a tensor algebra, what is the simplest presentation?

Tensor Element Decompositions

- **Main Problem**: Given an element in a tensor space, or a tensor algebra, what is the simplest presentation?
- Definition: Let U, V, W be finite-dimensional vector spaces over some field $\mathbb{K}$. A 3-tensor $T \in U \otimes V \otimes W$ is said to be of **rank one**, if $T = u \otimes v \otimes w$ for some $u \in U, v \in V, w \in W$. The **rank**, denoted $\mathrm{R}(T)$, of a tensor $T \in U \otimes V \otimes W$ is the smallest number r such that T may be expressed as a sum of r rank one tensors.

Notations of Tensor Elements

Notations of Tensor Elements

* Tensors may appear in various different guises.

Notations of Tensor Elements

* Tensors may appear in various different guises.
* 3-Tensors: $\sum_{i,j,k} a_{ijk}\, e_i \otimes f_j \otimes g_k$ with prescribed bases

$$\iff (a_{ijk})_{l\times m\times n} \iff f(x,y,z) = \sum_{i,j,k} a_{ijk} x_i y_j z_k$$

$$\iff \Phi : U \otimes V \otimes W \to \mathbb{K}, \quad e_i \otimes f_j \otimes g_k \mapsto a_{ijk}$$

Notations of Tensor Elements

Notations of Tensor Elements

* Call the tensor $\sum_{i_1,i_2,\ldots,i_d}^{n} a_{i_1 i_2 \ldots i_d}\, e_{i_1} \otimes e_{i_2} \otimes \cdots \otimes e_{i_d} \in V^{\otimes d}$ symmetric, if the constants $a_{i_1 i_2 \ldots i_d}$ are invariant under permuting the indices.

Notations of Tensor Elements

- Call the tensor $\sum_{i_1, i_2, \ldots, i_d}^{n} a_{i_1 i_2 \ldots i_d}\, e_{i_1} \otimes e_{i_2} \otimes \cdots \otimes e_{i_d} \in V^{\otimes d}$ symmetric, if the constants $a_{i_1 i_2 \ldots i_d}$ are invariant under permuting the indices.
- Symmetric tensors are synonymous with homogeneous polynomials, forms, symmetric multilinear forms, projective hypersurfaces, etc.

Notations of Tensor Elements

Notations of Tensor Elements

* For symmetric $T = \sum_{i_1, i_2, \ldots, i_d}^{n} a_{i_1 i_2 \ldots i_d} e_{i_1} \otimes e_{i_2} \otimes \cdots \otimes e_{i_d}$

$\iff T = v_1^{\otimes d} + v_2^{\otimes d} + \cdots + v_r^{\otimes d}$ for some $v_i \in V$

$\iff f(x_1, x_2, \ldots, x_n) = \sum_{i_1, i_2, \ldots, i_d}^{n} a_{i_1 i_2 \ldots i_d} x_{i_1} x_{i_2} \cdots x_{i_d} = \sum_{k=1}^{r} l_i^d$

where $l_i = \sum_{j=1}^{n} p_{ij} x_j \iff v_i = \sum_{j=1}^{n} p_{ij} e_j$ for all $1 \leq i \leq r$.

Notations of Tensor Elements

- For symmetric $T = \sum_{i_1,i_2,\ldots,i_d}^{n} a_{i_1 i_2 \ldots i_d}\, e_{i_1} \otimes e_{i_2} \otimes \cdots \otimes e_{i_d}$

$$\iff T = v_1^{\otimes d} + v_2^{\otimes d} + \cdots + v_r^{\otimes d} \text{ for some } v_i \in V$$

$$\iff f(x_1, x_2, \ldots, x_n) = \sum_{i_1,i_2,\ldots,i_d}^{n} a_{i_1 i_2 \ldots i_d} x_{i_1} x_{i_2} \cdots x_{i_d} = \sum_{k=1}^{r} l_i^d$$

where $l_i = \sum_{j=1}^{n} p_{ij} x_j \iff v_i = \sum_{j=1}^{n} p_{ij} e_j$ for all $1 \le i \le r$.

- The previous expression of f is called a Waring decomposition.

Tensor Element Decompositions

Tensor Element Decompositions

* Remark 1: It is extremely hard, in fact NP-hard, to determine the rank of a general high-order tensor.

Tensor Element Decompositions

* Remark 1: It is extremely hard, in fact NP-hard, to determine the rank of a general high-order tensor.

* Remark 2: For symmetric tensors, one can consider the problem of symmetric tensor rank. This is another version of the Waring problem of polynomials.

Tensor Element Decompositions

- Remark 1: It is extremely hard, in fact NP-hard, to determine the rank of a general high-order tensor.
- Remark 2: For symmetric tensors, one can consider the problem of symmetric tensor rank. This is another version of the Waring problem of polynomials.
- Remark 3: A rank one tensor is essentially a monomial in the more familiar terminology of algebra. It is nontrivial to determine if a polynomial is equivalent to a monomial.

Motivating Example 1: Sums of Squares

Motivating Example 1: Sums of Squares

* A beautiful sums of squares identity:
$$(x_1^2 + x_2^2)(y_1^2 + y_2^2) = (x_1y_1 + x_2y_2)^2 + (x_1y_2 - x_2y_1)^2 \,.$$

Motivating Example 1: Sums of Squares

* A beautiful sums of squares identity:
$$(x_1^2 + x_2^2)(y_1^2 + y_2^2) = (x_1y_1 + x_2y_2)^2 + (x_1y_2 - x_2y_1)^2 .$$

* For what values of n can there be identities like
$$(x_1^2 + \cdots + x_n^2)(y_1^2 + \cdots + y_n^2) = z_1^2 + \cdots + z_n^2$$
with $z_j \in \mathbb{K}[x_1, \ldots, x_n, y_1, \ldots, y_n]$?

Motivating Example 1: Sums of Squares

Motivating Example 1: Sums of Squares

* Hurwitz's "1, 2, 4, 8 Theorem" (1898):
  n-square identities exist only for n=1, 2, 4, 8.

Motivating Example 1: Sums of Squares

- **Hurwitz's "1, 2, 4, 8 Theorem" (1898):**
  n-square identities exist only for n=1, 2, 4, 8.

- **Hurwitz's Problem (1898):**
  Given r and s, find minimal n such that there holds
  $$(x_1^2 + \cdots + x_r^2)(y_1^2 + \cdots + y_s^2) = z_1^2 + \cdots + z_n^2 .$$

Motivating Example 1: Sums of Squares

Motivating Example 1: Sums of Squares

- **Hopf condition and Hopf's theorem:**
  $(x_1^2 + \cdots + x_r^2)(y_1^2 + \cdots + y_s^2) = z_1^2 + \cdots + z_n^2 \implies n \geq r * s$
  where $r * s$ is the Hopf-Stiefel function defined as
  $r * s := \min\{m \in \mathbb{N} \,|\, (x+y)^m = 0 \in \mathbb{F}_2[x,y]/(x^r, y^s)\}$.

Motivating Example 1: Sums of Squares

* **Hopf condition and Hopf's theorem:**
  $(x_1^2 + \cdots + x_r^2)(y_1^2 + \cdots + y_s^2) = z_1^2 + \cdots + z_n^2 \implies n \geq r * s$
  where $r * s$ is the Hopf-Stiefel function defined as
  $r * s := \min\{m \in \mathbb{N} \mid (x+y)^m = 0 \in \mathbb{F}_2[x,y]/(x^r, y^s)\}$.

* **The square identity induces a map of projective space** $\mathbb{P}^{r-1} \times \mathbb{P}^{s-1} \to \mathbb{P}^{n-1}$, **thus a map of the cohomology rings**
  $\mathbb{F}_2[T]/(T^n) \to \mathbb{F}_2[R]/(R^r) \otimes \mathbb{F}_2[S]/(S^s), \quad T \mapsto R \otimes 1 + 1 \otimes S$.
  Here, $\mathbb{F}_2[T]/(T^n)$ **is the cohomology ring of** $\mathbb{P}^{n-1}$ **. Then Hopf used the Hopf structures on the cohomology rings to deduce his condition. This is one of the historical origins of Hopf algebras!**

Motivating Example 1: Sums of Squares

Motivating Example 1: Sums of Squares

* On the other hand, the law of moduli of finite-dimensional real division algebras give rise to the 1, 2, 4, 8-square identities.

Motivating Example 1: Sums of Squares

* On the other hand, the law of moduli of finite-dimensional real division algebras give rise to the 1, 2, 4, 8-square identities.
* We noticed that the octonions are essentially a commutative associative algebra in suitable braided tensor categories. Then we could vastly generalize octonion algebras and applied them to generate square identities via partial law of moduli.

Motivating Example 1: Sums of Squares

* On the other hand, the law of moduli of finite-dimensional real division algebras give rise to the 1, 2, 4, 8-square identities.
* We noticed that the octonions are essentially a commutative associative algebra in suitable braided tensor categories. Then we could vastly generalize octonion algebras and applied them to generate square identities via partial law of moduli.
* Furthermore, it is natural to ask the composition law of general quadratic forms, or even higher degree forms, for example

$$(x_1^d + \cdots + x_r^d)(y_1^d + \cdots + y_s^d) = z_1^d + \cdots + z_n^d \, .$$

Motivating Example 2: Matrix Multiplication

Motivating Example 2: Matrix Multiplication

* Consider the multiplication of 2 by 2 matrices. As a bilinear map, it can be described as a 3-tensor:

$$M = \sum_{1 \leq i,j,k \leq 2} \epsilon_{ij} \otimes \epsilon_{jk} \otimes e_{ik} \, .$$

Motivating Example 2: Matrix Multiplication

* Consider the multiplication of 2 by 2 matrices. As a bilinear map, it can be described as a 3-tensor:

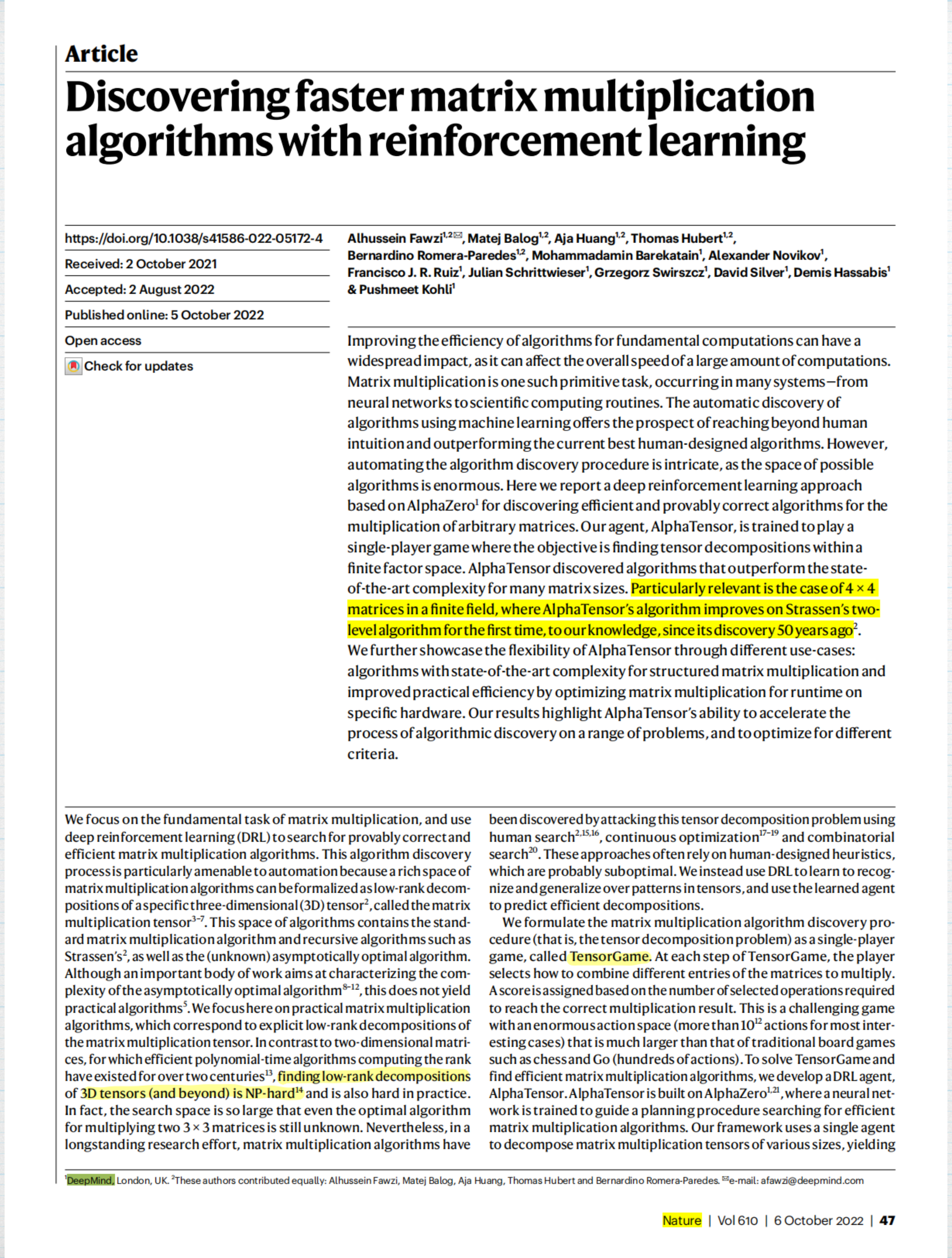
$$M = \sum_{1 \le i,j,k \le 2} \epsilon_{ij} \otimes \epsilon_{jk} \otimes e_{ik} \,.$$

* Strassen found that $\mathrm{R}(M) \le 7$! This implies that, it takes seven or even less scalar multiplications to compute the product of two 2 by 2 matrices. Therefore, it takes $n^{\log_2 7}$ scalar multiplications for that of n by n matrices.

Motivating Example 2: Matrix Multiplication

* The Strassen algorithm can be described as:

$$
\begin{aligned}
M = {} & (\epsilon_{11} + \epsilon_{22}) \otimes (\epsilon_{11} + \epsilon_{22}) \otimes (e_{11} + e_{22}) \\
& + (\epsilon_{21} + \epsilon_{22}) \otimes \epsilon_{11} \otimes (e_{21} - e_{22}) \\
& + \epsilon_{11} \otimes (\epsilon_{12} - \epsilon_{22}) \otimes (e_{12} + e_{22}) \\
& + \epsilon_{22} \otimes (\epsilon_{21} - \epsilon_{11}) \otimes (e_{11} + e_{21}) \\
& + (\epsilon_{11} + \epsilon_{12}) \otimes \epsilon_{22} \otimes (e_{12} - e_{11}) \\
& + (\epsilon_{21} - \epsilon_{11}) \otimes (\epsilon_{11} + \epsilon_{12}) \otimes e_{22} \\
& + (\epsilon_{12} - \epsilon_{22}) \otimes (\epsilon_{21} + \epsilon_{22}) \otimes e_{11} \, .
\end{aligned}
$$



Article

Discovering faster matrix multiplication algorithms with reinforcement learning

https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access

Check for updates

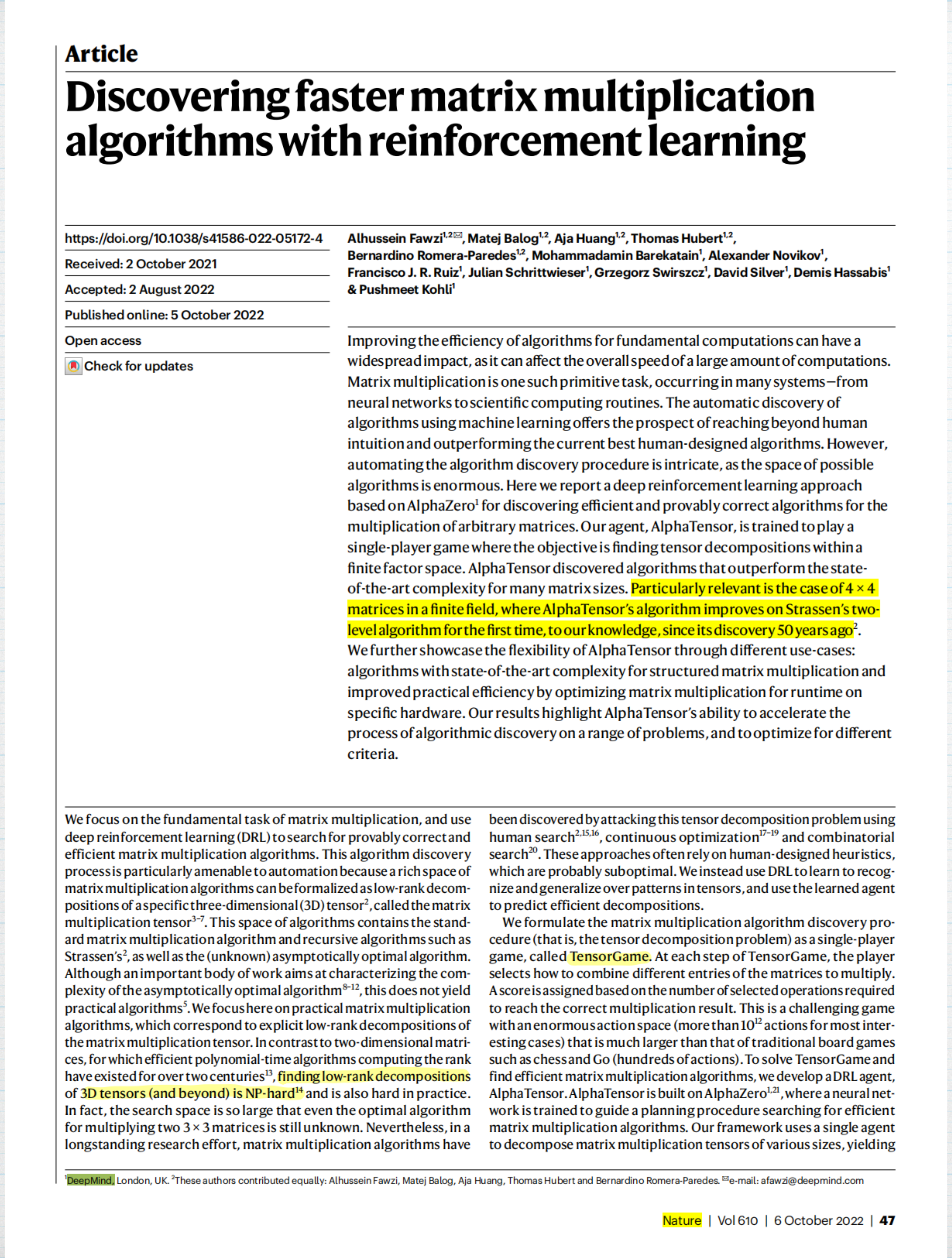
Alhussein Fawzi[1,2,✉], Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero[1] for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4 × 4 matrices in a finite field, where AlphaTensor's algorithm improves on Strassen's two-level algorithm for the first time, to our knowledge, since its discovery 50 years ago[2]. We further showcase the flexibility of AlphaTensor through different use-cases: algorithms with state-of-the-art complexity for structured matrix multiplication and improved practical efficiency by optimizing matrix multiplication for runtime on specific hardware. Our results highlight AlphaTensor's ability to accelerate the process of algorithmic discovery on a range of problems, and to optimize for different criteria.

We focus on the fundamental task of matrix multiplication, and use deep reinforcement learning (DRL) to search for provably correct and efficient matrix multiplication algorithms. This algorithm discovery process is particularly amenable to automation because a rich space of matrix multiplication algorithms can be formalized as low-rank decompositions of a specific three-dimensional (3D) tensor[2], called the matrix multiplication tensor[3–7]. This space of algorithms contains the standard matrix multiplication algorithm and recursive algorithms such as Strassen's[2], as well as the (unknown) asymptotically optimal algorithm. Although an important body of work aims at characterizing the complexity of the asymptotically optimal algorithm[8–12], this does not yield practical algorithms[5]. We focus here on practical matrix multiplication algorithms, which correspond to explicit low-rank decompositions of the matrix multiplication tensor. In contrast to two-dimensional matrices, for which efficient polynomial-time algorithms computing the rank have existed for over two centuries[13], finding low-rank decompositions of 3D tensors (and beyond) is NP-hard[14] and is also hard in practice. In fact, the search space is so large that even the optimal algorithm for multiplying two 3 × 3 matrices is still unknown. Nevertheless, in a longstanding research effort, matrix multiplication algorithms have been discovered by attacking this tensor decomposition problem using human search[2,15,16], continuous optimization[17–19] and combinatorial search[20]. These approaches often rely on human-designed heuristics, which are probably suboptimal. We instead use DRL to learn to recognize and generalize over patterns in tensors, and use the learned agent to predict efficient decompositions.

We formulate the matrix multiplication algorithm discovery procedure (that is, the tensor decomposition problem) as a single-player game, called TensorGame. At each step of TensorGame, the player selects how to combine different entries of the matrices to multiply. A score is assigned based on the number of selected operations required to reach the correct multiplication result. This is a challenging game with an enormous action space (more than 10^{12} actions for most interesting cases) that is much larger than that of traditional board games such as chess and Go (hundreds of actions). To solve TensorGame and find efficient matrix multiplication algorithms, we develop a DRL agent, AlphaTensor. AlphaTensor is built on AlphaZero[1,21], where a neural network is trained to guide a planning procedure searching for efficient matrix multiplication algorithms. Our framework uses a single agent to decompose matrix multiplication tensors of various sizes, yielding

[1]DeepMind, London, UK. [2]These authors contributed equally: Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert and Bernardino Romera-Paredes. ✉e-mail: afawzi@deepmind.com



Article

Discovering faster matrix multiplication algorithms with reinforcement learning

https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access

Check for updates

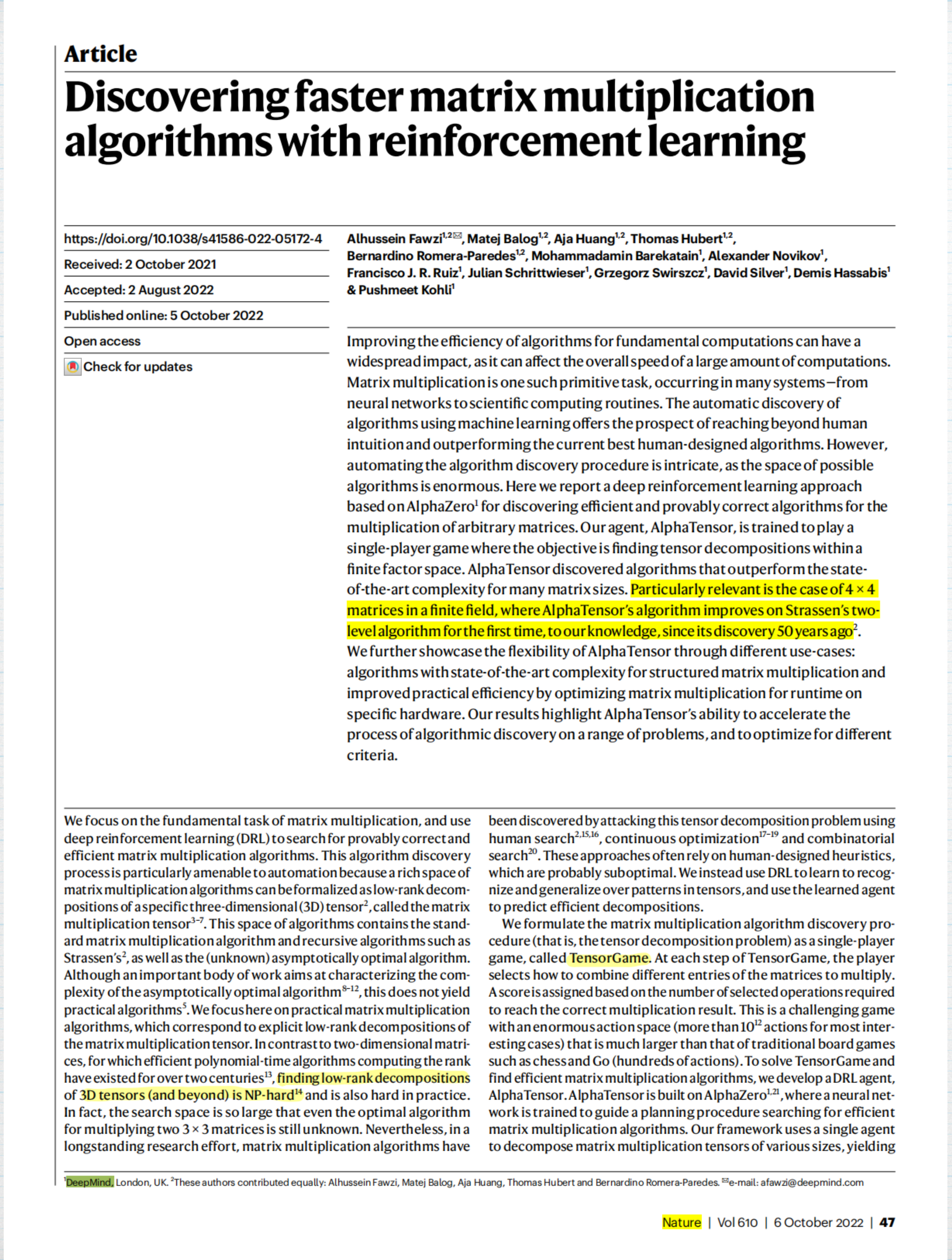
Alhussein Fawzi[1,2✉], Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero[1] for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4×4 matrices in a finite field, where AlphaTensor's algorithm improves on Strassen's two-level algorithm for the first time, to our knowledge, since its discovery 50 years ago[2]. We further showcase the flexibility of AlphaTensor through different use-cases: algorithms with state-of-the-art complexity for structured matrix multiplication and improved practical efficiency by optimizing matrix multiplication for runtime on specific hardware. Our results highlight AlphaTensor's ability to accelerate the process of algorithmic discovery on a range of problems, and to optimize for different criteria.

We focus on the fundamental task of matrix multiplication, and use deep reinforcement learning (DRL) to search for provably correct and efficient matrix multiplication algorithms. This algorithm discovery process is particularly amenable to automation because a rich space of matrix multiplication algorithms can be formalized as low-rank decompositions of a specific three-dimensional (3D) tensor[2], called the matrix multiplication tensor[3–7]. This space of algorithms contains the standard matrix multiplication algorithm and recursive algorithms such as Strassen's[2], as well as the (unknown) asymptotically optimal algorithm. Although an important body of work aims at characterizing the complexity of the asymptotically optimal algorithm[8–12], this does not yield practical algorithms[5]. We focus here on practical matrix multiplication algorithms, which correspond to explicit low-rank decompositions of the matrix multiplication tensor. In contrast to two-dimensional matrices, for which efficient polynomial-time algorithms computing the rank have existed for over two centuries[13], finding low-rank decompositions of 3D tensors (and beyond) is NP-hard[14] and is also hard in practice. In fact, the search space is so large that even the optimal algorithm for multiplying two 3×3 matrices is still unknown. Nevertheless, in a longstanding research effort, matrix multiplication algorithms have been discovered by attacking this tensor decomposition problem using human search[2,15,16], continuous optimization[17–19] and combinatorial search[20]. These approaches often rely on human-designed heuristics, which are probably suboptimal. We instead use DRL to learn to recognize and generalize over patterns in tensors, and use the learned agent to predict efficient decompositions.

We formulate the matrix multiplication algorithm discovery procedure (that is, the tensor decomposition problem) as a single-player game, called TensorGame. At each step of TensorGame, the player selects how to combine different entries of the matrices to multiply. A score is assigned based on the number of selected operations required to reach the correct multiplication result. This is a challenging game with an enormous action space (more than 10^{12} actions for most interesting cases) that is much larger than that of traditional board games such as chess and Go (hundreds of actions). To solve TensorGame and find efficient matrix multiplication algorithms, we develop a DRL agent, AlphaTensor. AlphaTensor is built on AlphaZero[1,21], where a neural network is trained to guide a planning procedure searching for efficient matrix multiplication algorithms. Our framework uses a single agent to decompose matrix multiplication tensors of various sizes, yielding

[1]DeepMind, London, UK. [2]These authors contributed equally: Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert and Bernardino Romera-Paredes. ✉e-mail: afawzi@deepmind.com



Article

Discovering faster matrix multiplication algorithms with reinforcement learning

https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access

Check for updates

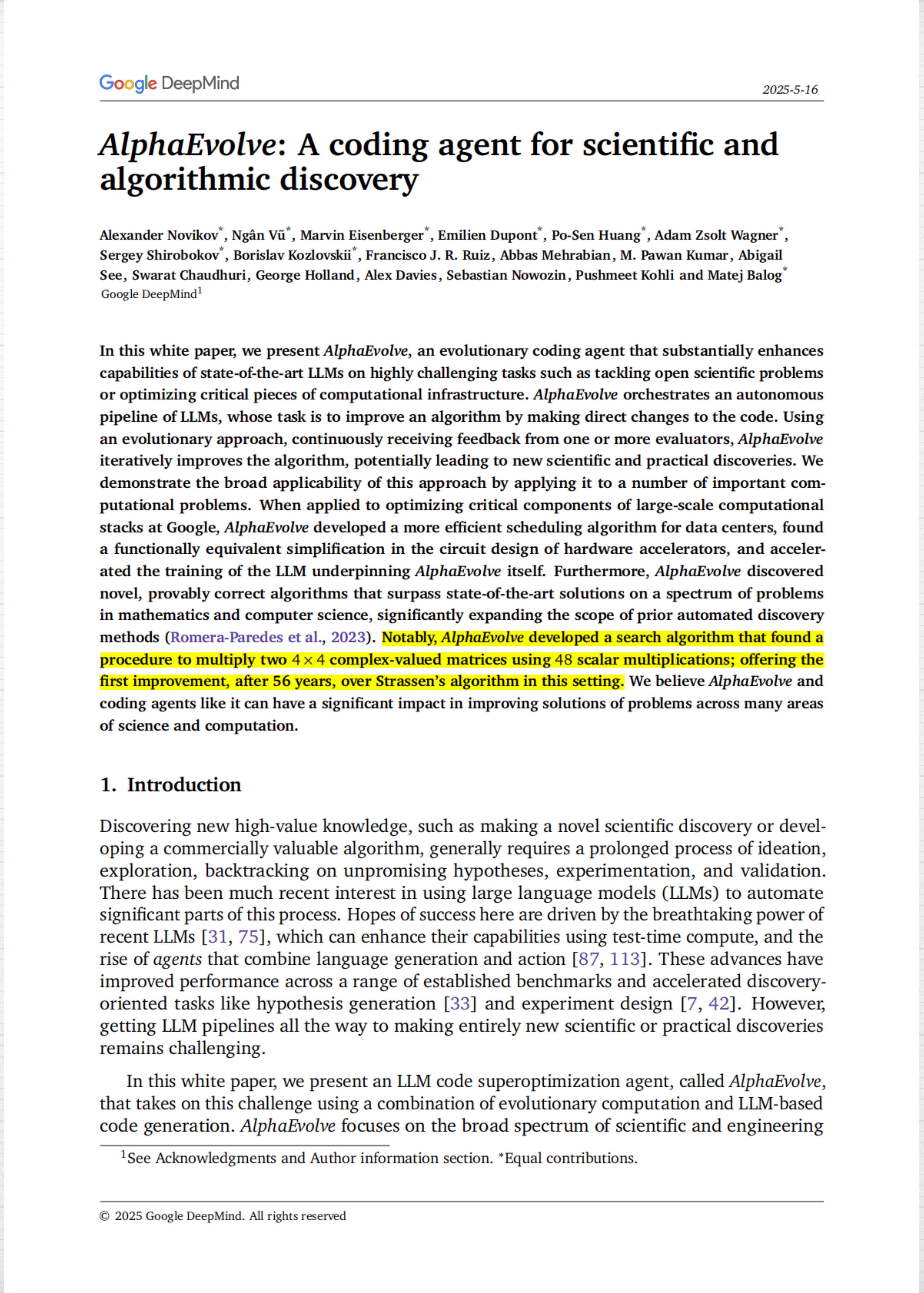
Alhussein Fawzi[1,2✉], Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero[1] for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4×4 matrices in a finite field, where AlphaTensor's algorithm improves on Strassen's two-level algorithm for the first time, to our knowledge, since its discovery 50 years ago[2]. We further showcase the flexibility of AlphaTensor through different use-cases: algorithms with state-of-the-art complexity for structured matrix multiplication and improved practical efficiency by optimizing matrix multiplication for runtime on specific hardware. Our results highlight AlphaTensor's ability to accelerate the process of algorithmic discovery on a range of problems, and to optimize for different criteria.

We focus on the fundamental task of matrix multiplication, and use deep reinforcement learning (DRL) to search for provably correct and efficient matrix multiplication algorithms. This algorithm discovery process is particularly amenable to automation because a rich space of matrix multiplication algorithms can be formalized as low-rank decompositions of a specific three-dimensional (3D) tensor[2], called the matrix multiplication tensor[3–7]. This space of algorithms contains the standard matrix multiplication algorithm and recursive algorithms such as Strassen's[2], as well as the (unknown) asymptotically optimal algorithm. Although an important body of work aims at characterizing the complexity of the asymptotically optimal algorithm[8–12], this does not yield practical algorithms[5]. We focus here on practical matrix multiplication algorithms, which correspond to explicit low-rank decompositions of the matrix multiplication tensor. In contrast to two-dimensional matrices, for which efficient polynomial-time algorithms computing the rank have existed for over two centuries[13], finding low-rank decompositions of 3D tensors (and beyond) is NP-hard[14] and is also hard in practice. In fact, the search space is so large that even the optimal algorithm for multiplying two 3×3 matrices is still unknown. Nevertheless, in a longstanding research effort, matrix multiplication algorithms have been discovered by attacking this tensor decomposition problem using human search[2,15,16], continuous optimization[17–19] and combinatorial search[20]. These approaches often rely on human-designed heuristics, which are probably suboptimal. We instead use DRL to learn to recognize and generalize over patterns in tensors, and use the learned agent to predict efficient decompositions.

We formulate the matrix multiplication algorithm discovery procedure (that is, the tensor decomposition problem) as a single-player game, called TensorGame. At each step of TensorGame, the player selects how to combine different entries of the matrices to multiply. A score is assigned based on the number of selected operations required to reach the correct multiplication result. This is a challenging game with an enormous action space (more than 10^{12} actions for most interesting cases) that is much larger than that of traditional board games such as chess and Go (hundreds of actions). To solve TensorGame and find efficient matrix multiplication algorithms, we develop a DRL agent, AlphaTensor. AlphaTensor is built on AlphaZero[1,21], where a neural network is trained to guide a planning procedure searching for efficient matrix multiplication algorithms. Our framework uses a single agent to decompose matrix multiplication tensors of various sizes, yielding

[1]DeepMind, London, UK. [2]These authors contributed equally: Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert and Bernardino Romera-Paredes. ✉e-mail: afawzi@deepmind.com



AlphaEvolve: A coding agent for scientific and algorithmic discovery

Alexander Novikov*, Ngân Vũ*, Marvin Eisenberger*, Emilien Dupont*, Po-Sen Huang*, Adam Zsolt Wagner*, Sergey Shirobokov*, Borislav Kozlovskii*, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli and Matej Balog*
Google DeepMind[1]

In this white paper, we present *AlphaEvolve*, an evolutionary coding agent that substantially enhances capabilities of state-of-the-art LLMs on highly challenging tasks such as tackling open scientific problems or optimizing critical pieces of computational infrastructure. *AlphaEvolve* orchestrates an autonomous pipeline of LLMs, whose task is to improve an algorithm by making direct changes to the code. Using an evolutionary approach, continuously receiving feedback from one or more evaluators, *AlphaEvolve* iteratively improves the algorithm, potentially leading to new scientific and practical discoveries. We demonstrate the broad applicability of this approach by applying it to a number of important computational problems. When applied to optimizing critical components of large-scale computational stacks at Google, *AlphaEvolve* developed a more efficient scheduling algorithm for data centers, found a functionally equivalent simplification in the circuit design of hardware accelerators, and accelerated the training of the LLM underpinning *AlphaEvolve* itself. Furthermore, *AlphaEvolve* discovered novel, provably correct algorithms that surpass state-of-the-art solutions on a spectrum of problems in mathematics and computer science, significantly expanding the scope of prior automated discovery methods (Romera-Paredes et al., 2023). Notably, *AlphaEvolve* developed a search algorithm that found a procedure to multiply two 4×4 complex-valued matrices using 48 scalar multiplications; offering the first improvement, after 56 years, over Strassen's algorithm in this setting. We believe *AlphaEvolve* and coding agents like it can have a significant impact in improving solutions of problems across many areas of science and computation.

1. Introduction

Discovering new high-value knowledge, such as making a novel scientific discovery or developing a commercially valuable algorithm, generally requires a prolonged process of ideation, exploration, backtracking on unpromising hypotheses, experimentation, and validation. There has been much recent interest in using large language models (LLMs) to automate significant parts of this process. Hopes of success here are driven by the breathtaking power of recent LLMs [31, 75], which can enhance their capabilities using test-time compute, and the rise of *agents* that combine language generation and action [87, 113]. These advances have improved performance across a range of established benchmarks and accelerated discovery-oriented tasks like hypothesis generation [33] and experiment design [7, 42]. However, getting LLM pipelines all the way to making entirely new scientific or practical discoveries remains challenging.

In this white paper, we present an LLM code superoptimization agent, called *AlphaEvolve*, that takes on this challenge using a combination of evolutionary computation and LLM-based code generation. *AlphaEvolve* focuses on the broad spectrum of scientific and engineering

[1]See Acknowledgments and Author information section. *Equal contributions.

What Do We Do?

What Do We Do?

* Problem 1: How to decompose a tensor as a direct sum, or equivalently, a partial decomposition?

What Do We Do?

- Problem 1: How to decompose a tensor as a direct sum, or equivalently, a partial decomposition?
- Problem 2: How to decompose a tensor as a direct product, i.e., a multiplicative decomposition?

What Do We Do?

* Problem 1: How to decompose a tensor as a direct sum, or equivalently, a partial decomposition?
* Problem 2: How to decompose a tensor as a direct product, i.e., a multiplicative decomposition?
* Problem 3: Determine if a tensor is of rank one, or if a homogeneous polynomial is a product of linear forms.

Direct Sum

Direct Sum

* A tensor $T \in U \otimes V \otimes W$ is called a direct sum, if there are $T_i \in U_i \otimes V_i \otimes W_i \quad (i = 1,2)$ such that $T = T_1 + T_2$ in the tensor space $(U_1 \oplus U_2) \otimes (V_1 \oplus V_2) \otimes (W_1 \oplus W_2) = U \otimes V \otimes W$.

Direct Sum

* A tensor $T \in U \otimes V \otimes W$ is called a direct sum, if there are $T_i \in U_i \otimes V_i \otimes W_i \quad (i = 1,2)$ such that $T = T_1 + T_2$ in the tensor space $(U_1 \oplus U_2) \otimes (V_1 \oplus V_2) \otimes (W_1 \oplus W_2) = U \otimes V \otimes W$.

* In terms of polynomials, this is equivalent to writing $f(X, Y, Z) = g(X_1, Y_1, Z_1) + h(X_2, Y_2, Z_2)$ where $X = X_1 \sqcup X_2$, etc. In other words, separate variables of polynomials.

Strassen's Additivity Conjecture

Strassen's Additivity Conjecture

* Multiplication of blocked matrices led Strassen to propose his additivity conjecture:
  Given $T_i \in U_i \otimes V_i \otimes W_i \quad (i = 1,2)$, let $T = T_1 \oplus T_2$ in the tensor space $(U_1 \oplus U_2) \otimes (V_1 \oplus V_2) \otimes (W_1 \oplus W_2)$.
  Then $$\mathrm{R}(T) = \mathrm{R}(T_1) + \mathrm{R}(T_2).$$

Strassen's Additivity Conjecture

* Multiplication of blocked matrices led Strassen to propose his additivity conjecture:
  Given $T_i \in U_i \otimes V_i \otimes W_i \quad (i = 1,2)$, let $T = T_1 \oplus T_2$ in the tensor space $(U_1 \oplus U_2) \otimes (V_1 \oplus V_2) \otimes (W_1 \oplus W_2)$.
  Then $$\mathrm{R}(T) = \mathrm{R}(T_1) + \mathrm{R}(T_2).$$

* The conjecture is disproved by Shitov (Acta Math. 2019). It seems that the conjecture fails only in very strange situations. It is of interest to consider the conditions for "=" holds.

Direct Product

* Definition: A multivariate function $f(x_1, x_2, \ldots, x_n)$ is called a direct product if, after a linear change of variables, it can be written as a product of two or more functions in disjoint sets of variables

$$f(x) \overset{x=Py}{=} g_1(y_1, \ldots, y_a) \cdot g_2(y_{a+1}, \ldots, y_n), \quad 0 < a < n\,.$$

Call $f(x_1, x_2, \ldots, x_n)$ completely factorable, if it is a product of n univariate functions.

Direct Sum vs Direct Product

* **Lemma.** (1) $f(x_1, x_2, \ldots, x_n)$ is a direct sum if and only if $\exp^f$ is a direct product. (2) $g(x_1, x_2, \ldots, x_n)$ is a direct product if and only if $\log g$ is a direct sum.

Center and Decompositions

* Definition: The center of an n-variate function $f(x_1, x_2, \ldots, x_n)$ is defined as

$$Z(f) := \{X \in \mathbb{K}^{n\times n} \mid (HX)^T = HX\},$$

where $H = \left(\dfrac{\partial^2 f}{\partial x_i \partial x_j}\right)_{1\leq i,j\leq n}$ is the hessian matrix of the function $f(x_1, x_2, \ldots, x_n)$.

Center and Decompositions

* Proposition: (1) The center of a multivariate function is a special Jordan algebra. (2) Direct sum decompositions of f are in bijection with complete sets of orthogonal idempotent of $Z(f)$. (3) Direct product decompositions of f are in bijection with complete sets of orthogonal idempotents of $Z(\log f)$.

Toy Example 1

Toy Example 1

* If $f = x^3 + y^3$, then its center $Z(f) \cong \mathbb{K}^2$.

Toy Example 1

- If $f = x^3 + y^3$, then its center $Z(f) \cong \mathbb{K}^2$.
- General binary cubics: $a_0x^3 + 3a_1x^2y + 3a_2xy^2 + a_3y^3$

Toy Example 1

* If $f = x^3 + y^3$, then its center $Z(f) \cong \mathbb{K}^2$.
* General binary cubics: $a_0x^3 + 3a_1x^2y + 3a_2xy^2 + a_3y^3$
* By dimension counting:
  $a_0x^3 + 3a_1x^2y + 3a_2xy^2 + a_3y^3 = (\alpha_1x + \beta_1y)^3 + (\alpha_2x + \beta_2y)^3$

Toy Example 1

* If $f = x^3 + y^3$, then its center $Z(f) \cong \mathbb{K}^2$.
* General binary cubics: $a_0x^3 + 3a_1x^2y + 3a_2xy^2 + a_3y^3$
* By dimension counting:
  $a_0x^3 + 3a_1x^2y + 3a_2xy^2 + a_3y^3 = (\alpha_1x + \beta_1y)^3 + (\alpha_2x + \beta_2y)^3$
* Cardano formula revisited:
  $a_0x^3 + 3a_1x^2 + 3a_2x + a_3 = (\alpha_1x + \beta_1)^3 + (\alpha_2x + \beta_2)^3$

Toy Example 2

Toy Example 2

* Consider the following ternary cubic

$$f(x_1, x_2, x_3) = x_1^3 + x_2^3 + x_3^3 - 3x_1x_2x_3\,.$$

Toy Example 2

* Consider the following ternary cubic
$$f(x_1, x_2, x_3) = x_1^3 + x_2^3 + x_3^3 - 3x_1x_2x_3\,.$$

* The center of the function $\log f$ is
$$Z(\log f) = \mathbb{C}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \mathbb{C}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} + \mathbb{C}\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Toy Example 2

Toy Example 2

* It is easy to find three mutually orthogonal idempotents in $Z(\log f)$ which correspond to a complete factorization of the polynomial $f(x_1, x_2, x_3)$.

Toy Example 2

- It is easy to find three mutually orthogonal idempotents in $Z(\log f)$ which correspond to a complete factorization of the polynomial $f(x_1, x_2, x_3)$.

- The polynomial f is factorized as
  $f(x_1, x_2, x_3) = (x_1 + x_2 + x_3)(x_1 + \omega x_2 + \omega^2 x_3)(x_1 + \omega^2 x_2 + \omega x_3)$
  where $\omega = \dfrac{-1 + \sqrt{3}i}{2}$ is a primitive cubic root of unity.

The Chow Variety

The Chow Variety

* Definition: The Chow variety of product of linear forms is the orbit closure $\overline{\mathrm{GL}_n[x_1x_2\cdots x_n]}$.

The Chow Variety

* Definition: The Chow variety of product of linear forms is the orbit closure $\overline{\mathrm{GL}_n[x_1x_2\cdots x_n]}$.

* Problem: determine whether a higher degree form is a product of linear forms.

The Chow Variety

The Chow Variety

- **Theorem:** Suppose $f \in \mathbb{K}[x_1, x_2, \ldots, x_n]_d$. Then $f = l_1^{d_1} l_2^{d_2} \ldots l_n^{d_n}$ $(d_i \geq 1, \forall i)$ with linearly independent l_i's if and only if $Z(\log f) \cong \mathbb{K}^n$.

The Chow Variety

- **Theorem:** Suppose $f \in \mathbb{K}[x_1, x_2, \ldots, x_n]_d$. Then $f = l_1^{d_1} l_2^{d_2} \ldots l_n^{d_n}$ $(d_i \geq 1, \forall i)$ with linearly independent l_i's if and only if $Z(\log f) \cong \mathbb{K}^n$.
- **Remark:** Our algorithm of direct product gives a simple criterion and an efficient algorithm to completely factorize a multivariate polynomial.

Toy Example 3

Toy Example 3

* Consider the quartic $x^4 + y^4 + (x+y)^4$. How do we recognize the terms of powers of linear forms?

Toy Example 3

- Consider the quartic $x^4 + y^4 + (x+y)^4$. How do we recognize the terms of powers of linear forms?
- By calculation, it is easy to observe that $\partial_x \partial_y (\partial_x - \partial_y)$ annihilates the previous quartic. Moreover, each linear factor of the annihilating differential polynomial corresponds to a term of the Waring decomposition!

Hopf Pairing vs The Apolarity Method

Hopf Pairing vs The Apolarity Method

* Denote $\mathscr{R} = \mathbb{K}[x_1, x_2, \ldots, x_n]$ and $\mathscr{D} = \mathbb{K}[\partial_1, \partial_2, \ldots, \partial_n]$. Then $\mathscr{D}$ acts naturally as differentiation on $\mathscr{R}$.

Hopf Pairing vs The Apolarity Method

* Denote $\mathscr{R} = \mathbb{K}[x_1, x_2, \ldots, x_n]$ and $\mathscr{D} = \mathbb{K}[\partial_1, \partial_2, \ldots, \partial_n]$. Then $\mathscr{D}$ acts naturally as differentiation on $\mathscr{R}$.

* $\mathscr{R}$ and $\mathscr{D}$ are Hopf algebras, and they are mutually graded dual of each other. The previous action comes from Hopf algebraic structures and it provides a nice Hopf pairing.

Hopf Pairing vs The Apolarity Method

- Denote $\mathscr{R} = \mathbb{K}[x_1, x_2, \ldots, x_n]$ and $\mathscr{D} = \mathbb{K}[\partial_1, \partial_2, \ldots, \partial_n]$. Then $\mathscr{D}$ acts naturally as differentiation on $\mathscr{R}$.
- $\mathscr{R}$ and $\mathscr{D}$ are Hopf algebras, and they are mutually graded dual of each other. The previous action comes from Hopf algebraic structures and it provides a nice Hopf pairing.
- Rota highly advocated the Hopf algebraic approach to the apolarity method and even the whole classical algebraic invariant theory!

Hopf Pairing vs The Apolarity Method

Hopf Pairing vs The Apolarity Method

* For $f \in \mathscr{R}_d$, let $f^\perp = \{g \in \mathscr{D} \mid g \circ f = 0\}$. Then $f^\perp$ is a Gorenstein ideal and $\mathscr{D}/f^\perp$ is a Gorenstein algebra. A well known result of Macaulay says that: there is a bijection between forms and graded Atinian Gorenstein algebras.

Hopf Pairing vs The Apolarity Method

- For $f \in \mathscr{R}_d$, let $f^\perp = \{g \in \mathscr{D} \mid g \circ f = 0\}$. Then $f^\perp$ is a Gorenstein ideal and $\mathscr{D}/f^\perp$ is a Gorenstein algebra. A well known result of Macaulay says that: there is a bijection between forms and graded Atinian Gorenstein algebras.

- If $g \in \mathscr{D}_d$, then $g^\perp = \{f \in \mathscr{R} \mid g \circ f = 0\} \subset \mathscr{R}$ is a coideal. Determining $g^\perp$ amounts to solving a constant coefficient PDE! For example, consider the Laplacian $\Delta = \partial_{x_1}^2 + \ldots + \partial_{x_n}^2$, then $\Delta^\perp$ consists of the well-known harmonic polynomials.

Hopf Pairing vs The Apolarity Method

Hopf Pairing vs The Apolarity Method

* The Apolarity Lemma: Suppose $f \in \mathbb{C}[x_1, \ldots, x_n]$. Then

$$f = \sum_{i=1}^{r} \lambda_i(\alpha_{i1}x_1 + \ldots + \alpha_{in}x_n)^d \iff \bigcap_{i=1}^{r} I_{[\alpha_{i1} \,:\, \cdots \,:\, \alpha_{in}]} \subset f^{\perp}\,.$$

Hopf Pairing vs The Apolarity Method

- The Apolarity Lemma: Suppose $f \in \mathbb{C}[x_1, \ldots, x_n]$. Then

$$f = \sum_{i=1}^{r} \lambda_i(\alpha_{i1}x_1 + \ldots + \alpha_{in}x_n)^d \iff \bigcap_{i=1}^{r} I_{[\alpha_{i1} \,:\, \cdots \,:\, \alpha_{in}]} \subset f^{\perp}.$$

- For binary forms, we provide an elementary treatment. For forms with more variables, it seems Hopf pairings might be a promising approach.

A Brief Summary

A Brief Summary

* The problem of Tensor decompositions arises naturally form arithmetic, algebra, geometry, multilinear algebra, data processing, and various other practical areas.

A Brief Summary

* The problem of Tensor decompositions arises naturally form arithmetic, algebra, geometry, multilinear algebra, data processing, and various other practical areas.

* There remain many challenging problems. Methods from Hopf algebras and tensor categories, representation theory, invariant theory, algebraic geometry, and even machine learning, artificial intelligence, etc. play significant roles for further studying.

Thank You!

Thank You!

Questions or Comments?