[Paper category here]

Discovering faster matrix multiplication algorithms with reinforcement learning

Authors: Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis & Pushmeet Kohli
Affiliation: DeepMind, London, UK.
Publication: Nature
Year: 5 Oct. 2022
Link: https://www.nature.com/articles/s41586-022-05172-4

---

1. Summary

Problem

- The paper addresses the challenge of **automating the discovery of efficient and provably correct algorithms for matrix multiplication**.
- Matrix multiplication is a fundamental computational primitive used across many systems, from neural networks to scientific computing routines. Improving the efficiency of this can **accelerate a massive amount of computations**.

Method

- Core idea: To frame the discovery of practical matrix multiplication algorithms as finding **low-rank decompositions of a 3D matrix multiplication tensor**.
    - **TensorGame**: A single-player game that formulates the tensor decomposition problem.
    - **AlphaTensor**: An agent that solves TensorGame and finds efficient matrix multiplication algorithms.
    - TensorGame is played as follows:
        - State $S_0 = T_n$
        - Each move is a triplet (u, v, w) and the environment subtracts $u \otimes v \otimes w$.
        - Reaching the zero tensor in R moves is a guaranteed exact rank-R algorithm.
        - Reward -1 per step makes the shortest game the best algorithm.

- Novel technique(s)
    - **Transformer-based architecture with multi-grid axial attention**: Exploits the symmetries of 3D tensors by projecting them into three grids and applying axial attention, which is invariant to slice reordering.
    - **Synthetic demonstrations**: Training uses a mix of standard RL loss and supervised learning on randomly generated tensor-factorization pairs to jump-start the agent.
    - **Change of basis augmentation**: Generating diverse game states by expressing the target tensor in random bases, which allows the agent to explore the rich algorithm space more effectively.

Main Contributions

- **Discovered novel matrix multiplication algorithms that outperform the state-of-the-art complexity** for many matrix sizes, including a new algorithm for 4x4 matrices in a finite field that beats Strassen's two-level algorithm.
- Demonstrated that AlphaTensor can be **optimized for practical hardware runtime**, discovering tailored algorithms that run significantly faster on specific GPUs and TPUs than standard fast algorithms.
- Generated a large database containing up to thousands of diverse, non-equivalent algorithms for each matrix size, proving that **the space of matrix multiplication algorithms is much richer than previously thought**.

Experimental Setup

- Task: Decomposing a 3D target tensor into a sum of rank-one tensors in the minimum number of steps.
- Dataset(s): A pre-generated 5M synthetic tensor-factorization pairs.
- Baseline(s):
    - **For theoretical complexity**: Strassen (1969), Laderman (1976), Hopcroft and Kerr (1971), Smirnov (2013), and Sedoglavic and Smirnov (2021)
    - **For practical runtime**: standard matrix multiplication like cuBLAS and the Strassen-square algorithm.
- Evaluation Metric(s)
    - **Tensor rank**: the theoretical number of scalar multiplications required
    - **speed-up percentage**: practical runtime on target hardware

Experimental Results

- Best result: AlphaTensor discovered a method to multiply 4×4 matrices in modular arithmetic using **47 multiplications**, and a method for 4x5 by 5x5 matrices in standard arithmetic using **76 multiplications.**
- Comparison to baselines: The agent matched or improved over the state-of-the-art theoretical rank bounds for more than 70 matrix multiplication sizes. In hardware benchmarking, it yielded performance speed-ups of up to **23.9% on an NVIDIA V100 GPU** and **13.9% on a TPU v2** compared to the Strassen-square algorithm.
- Key takeaway: Deep Reinforcement Learning can successfully automate algorithmic discovery for core mathematical problems, surpassing human intuition to find both computationally optimal and hardware-tailored solutions.

2. Background & Motivation

Problem Statement

- Previous research mathematically formalized matrix multiplication as finding explicit low-rank decompositions of a specific 3D matrix multiplication tensor.
- This paper translates this NP-hard problem into a single-player game, **TensorGame**, where an agent iteratively selects factors to subtract from the tensor.

Why is this problem important?

- Because the search space for tensor decomposition is large ,over 10^{12} actions for interesting cases, finding the optimal algorithm is NP-hard.
- Consequently, even the optimal algorithm for multiplying two 3x3 matrices remains unknown.

Limitations of Previous Work

- Human search relies on suboptimal heuristics.
- Continuous optimization yields approximate floating-point solutions requiring manual intervention to extract exact algorithms.
- SAT solvers cannot scale to larger tensors due to explosive combinatorial growth.

3. Related Work

Existing Approaches

- Manual mathematical derivation: Strassen, Laderman
- Continuous optimization methods: alternating least squares
- Supervised learning with continuous networks
- Combinatorial solvers: Boolean satisfiability SAT formulations for 3x3

Differences from Prior Work

- AlphaTensor outputs exact, discrete algorithms without quantization, scales to massive spaces using sample-based MCTS, and discovers optimal algorithms completely from scratch without needing initial reference code.

4. Method

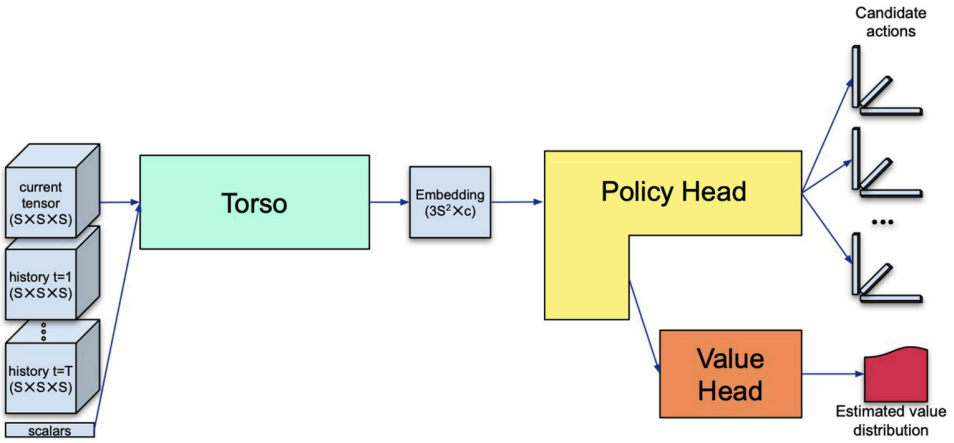
Overview

- The **AlphaTensor** is built on a closed-loop system that combines a deep neural network with sample-based **Monte Carlo Tree Search** to discover exact matrix multiplication algorithms.
- This framework operates by casting algorithmic discovery as a single-player game, **TensorGame**.
- The system consists of two primary loops:
    - **Acting**: generating games via MCTS planning
    - **Learning**: updating the neural network

Pipeline

1. **Initialization**: Target tensor is set as state $S0$, optionally modified by a random change of basis.
2. **MCTS Planning**: The network guides search by predicting action policies and value return distributions.
3. **State Update**: An action, triplet factor subtraction, is applied to transition to the next state.
4. **Termination**: The game ends at the zero tensor, outputting a mathematically verified algorithm.

Model Architecture



- The model consists of a **torso** that processes states and history, the sequence of the last h actions (moves) made by the player in the current game, followed by a **policy head** and a **value head**:
  - **Input processing**: The network receives the current 3D tensor S_t ($S \times S \times S$), the history of the last h actions ($h=7$) concatenated as rank-1 tensors, and a list of scalars including the current time index t.
  - **The Torso**: Operates over three $S \times S$ grids projected from the input tensor along its three modes. Each grid element is a feature vector enriched by concatenating a projection from the input scalars and mapping them to a 512-dimensional space. The grids then pass through a sequence of attention-based blocks containing multi-grid axial attention layers.
  - **The Policy Head**: A causal transformer model that autoregressively predicts tokenized representations of the three factor vectors (u,v,w).
  - **The Value Head**: A 4-layer MLP that outputs quantile predictions to represent the return distribution.

Key Components

Component	Purpose
Torso	Maps the 3D tensor state and scalar inputs to a 512-dimensional feature space by performing axial attention in parallel across pairs of grids, which respects the tensor rank's invariance to slice reordering.
Policy Head	Predicts a distribution over the massive discrete action space by decomposing the factors into tokens and cross-attending to the torso features.
Value Head	Estimates the return distribution using quantile regression, using the average of the top 25% quantiles to guide the MCTS with a risk-seeking evaluation metric.
Synthetic Demonstrations	Bootstraps the agent with 5 million randomly generated tensor-factorization pairs to jump-start search in the NP-hard algorithm space.
Change of Basis	Transforms the target tensor randomly during training, injecting massive game play diversity to prevent the agent from getting stuck in local minima.
Action Canonicalization	Forces target actions into a canonical form during training to avoid wasting network capacity on mathematically equivalent vector sign permutations.

Loss Functions

- KL divergence loss: to update the policy head, aligning its predictions with the empirical search policy or target demonstrations
- quantile regression distributional loss: to train the value head

Training Strategy

- Multi-target training across different tensor sizes on 64 TPU v3 cores with a total batch size of 2,048. Game generation is handled in parallel by 1,600 standalone TPU v4 actors running MCTS.

5. Experiments

Datasets

- **Synthetic Demonstrations**
    - 5 million random tensor-factorization pairs, filtered to remove suboptimal entries where $u,v,\text{or } w=0$.
- **Self-Play**
    - Trajectories of self-play games.
- **High-Score Reinforcement**
    - Discovered games with high scores are added to the demonstration buffer.

Experimental Settings

- **Targets:** Rectangular and square matrix multiplications up to size 5×5, zero-padded to $T5$ $(25\times25\times25)$.
- **Fields:** Standard arithmetic (R) with coefficients $F=\{-2,...,2\}$ and Modular arithmetic $(\mathrm{Z}2)$ with coefficients $F=\{0,1\}$.
- **Changes of Basis:** 100,000 random bases generated via unimodular matrices where $A=B=C$ and $\det A\in\{-1,+1\}$.
- **Data Augmentations:** Swapping random actions with the final action, and applying 100 pre-sampled signed permutations.
- **Hardware Settings:** Benchmark on Nvidia V100 GPU and Google TPU v2 for 8,192x8,192 square multiplications (split into 2,048 x 2,048 blocks, no recursion) compiled to JAX.

Evaluation Metrics

- **Tensor Rank***:* The step count R to reach $SR=0$.
- **Terminal Penalty***:* Sum of the matrix ranks of terminal tensor slices if step limit $Rlimit$ is reached.
- **Hardware Speed-up***:* Median runtime speed-up (%) over 200 runs compared to standard libraries (e.g., cuBLAS) and Strassen-square.
- **MCTS Risk-Seeking Value***:* Guiding search using the maximum possible return rather than the mean.

Baseline Methods

- **Theory**: Strassen, Laderman, Hopcroft & Kerr, Smirnov, and Sedoglavic & Smirnov
- **Practical**: cuBLAS (standard routines) and Strassen-square.

6. Results & Analysis

Main Results

- **Theoretical bounds**: Matched or improved over state-of-the-art rank bounds for more than 70 tensors of sizes up to 12.
- **Modular breakthrough**: Discovered a 47-multiplication algorithm for 4×4 matrices in Z_2, improving on Strassen's two-level 49-multiplication method, which recursively achieves an asymptotic complexity of $O(N^{2.778})$.
- **Standard arithmetic**: Discovered a rank-76 decomposition for the $4\times5\times5$ tensor, outperforming the previous state of the art of 80 multiplications.
- **Hardware execution**: Achieved median runtime speedups of up to 23.9% on Nvidia V100 GPUs and 13.9% on Google TPU v2 over standard compiler implementations.

Ablation Studies

- **Training Mix**: Training on a hybrid combination of target tensors and random synthetic demonstrations substantially outperformed training on either target tensors or synthetic demonstrations alone, despite the random tensors possessing different mathematical properties.
- **Multi-Target and Transfer**: Training a single agent to decompose multiple target tensors simultaneously facilitated the direct transfer of mathematical strategies across sizes and arithmetics, improving overall agent performance over isolated single-target training.
- **Hardware Tailoring**: Algorithms specifically optimized for one physical device performed poorly when run on another. For instance, the TPU-optimized algorithm yielded a 10.3% speedup on TPU but fell to a 4.4% speedup on GPU, while the GPU-optimized algorithm yielded an 8.5% speedup on GPU but only 2.8% on TPU.

Qualitative Analysis

- **Decomposition Symmetries**: AlphaTensor proved that the algorithm space is vastly richer than previously assumed by discovering over 14,000 non-equivalent, mathematically unique rank-49 factorizations for $T4$ in standard arithmetic. These factorizations cannot be mapped to one another via symmetry permutations and exhibit highly diverse properties in sparsity and matrix rank.
- **Generalization to Novel Operations**: The agent generalized and discovered structured mathematical patterns. For skew-symmetric matrix-vector products of size n, it discovered a general, asymptotically optimal algorithm using $n(n-1)(n+2)/2{\sim}n^3/2$ multiplications, outperforming prior n^2 algorithms.
- **Cyclic Convolution (DFT)**: When applied to cyclic convolutions in the finite field of order 17, the agent automatically re-discovered the optimal rank-n Discrete Fourier Transform (DFT) and inverse DFT matrix coefficients completely from scratch.

Failure Cases / Error Analysis

- **Predefined Coefficient Failure**: Discretizing the search space with a hardcoded coefficient set F, such as $\{-2,...,2\}$, prevents mathematical overflow but represents a major failure mode: the agent is completely unable to discover highly efficient algorithms that require coefficients outside of F.
- **Move Limit Termination**: If the agent fails to reach the zero tensor within the maximum step limit R_{limit}, the game terminates in an incomplete decomposition. Rather than finding a valid factorization, the system must apply a terminal penalty calculated using a loose mathematical upper bound on the residual tensor's rank.
- **Cross-Hardware Degradation**: Algorithms tailored to optimize physical execution times are highly platform-specific; running a GPU-tailored factorization on a TPU causes significant performance drops due to compiler-stack mismatch.

7. Discussion

Strengths

- **Theoretical & Mathematical Correctness**: Operates natively in discrete spaces to yield exact, provably correct mathematical algorithms, completely bypassing the floating-point errors of continuous optimization.
- **Unrivaled Search Scalability**: Effectively handles massive search spaces like T_4, which has an action space 10^{10} times larger than T_3, that are computationally unreachable for combinatorial SAT solvers.
- **Flexibility and Custom Objectives**: Supports complex, stochastic, and non-differentiable rewards, enabling direct optimization for physical on-chip runtime, arbitrary finite fields, and alternative structures.
- **Algorithmic Discovery From Scratch**: Discovers optimal mathematical operations entirely from scratch without requiring existing reference implementations or baseline human code.

Limitations

- **Discretized Action Constraints**: Performance is strictly constrained by the user-defined, hardcoded, finite coefficient set F.
- **High Computational Cost**: Training a converging agent requires an extensive footprint of physical hardware, taking about a week using 64 TPU v3 cores and 1,600 parallel TPU v4 actors.

Future Work

- **Searching for the Coefficient Set (F)**: A primary limitation of the current system is the need for human researchers to pre-define the allowed set of coefficients (F). Future research will focus on adapting AlphaTensor to **automatically search for and construct the optimal coefficient set** F, which could unlock even more efficient algorithms that require values outside the pre-defined integers.
- **Optimizing Alternative Metrics**: Because AlphaTensor's reinforcement learning reward scheme is highly flexible and can handle non-differentiable objectives, future work can optimize for physical and mathematical properties beyond rank and execution time, such as **numerical stability** or **on-chip energy consumption**.
- **Tackling Broader Mathematical Challenges**: The core methodology can be extended to solve other NP-hard mathematical primitives. This includes computing alternative mathematical formulations of rank, such as **border rank**, as well as solving complex matrix factorization problems like **non-negative factorization**.
- **Guiding Pure Mathematical Research**: The database of thousands of diverse, non-equivalent algorithms discovered by AlphaTensor provides mathematicians with a rich, empirical dataset to analyze. Studying the symmetries of these factorizations is expected to guide theoretical research into the **asymptotic complexity of matrix multiplication**.

8. Personal Insights

Key Takeaways

- AlphaTensor demonstrates that deep reinforcement learning can crack open mathematical problems that resisted human efforts for half a century.
- It highlights that theoretical simplicity does not always equal practical speed; optimizing for hardware compilation pipelines can unlock substantial execution efficiency.

Ideas Relevant to the Saegyeol AI Project

- **Sequential Security Routing as a Guided Planning Game**: The multi-step orchestration of a research workflow—from classifying sensitivity and masking variables to choosing local or cloud execution paths—can be modeled as a sequential decision-making game. This directly applies to the paper's framework of **TensorGame**, which translates abstract matrix decompositions into a step-by-step

subtraction game navigated via Monte Carlo Tree Search (MCTS). By guiding this search with a neural network and a **risk-seeking value function** focused on finding the single best mathematical trajectory, SaeGyeol's Private AI Control Plane can dynamically plan and select the absolute safest, most cost-effective execution path for highly complex, multi-stage scientific queries.

- **Cold-Start Routing via "Reverse" Synthetic Generation**: To train SaeGyeol's routing classifiers before any actual, highly sensitive institutional data is available, the Control Plane can generate high-quality training datasets **"in reverse"**. This mirrors AlphaTensor's synthetic bootstrapping strategy, which overcomes the NP-hard nature of tensor decomposition by first sampling random factor triplets, multiplying them to construct synthetic target tensors, and filtering out suboptimal zero-valued entries to build a robust dataset of 5 million demonstrations. By defining compliant execution paths and variable masking pipelines first, SaeGyeol can synthetically reconstruct corresponding dummy research queries and metadata schemas to pre-train and validate its routing networks on day one.
- **Sovereign Compute Protection via Mathematical "Change of Basis"**: When researchers need to run matrix operations on highly sensitive, proprietary arrays (such as proprietary genomic or molecular datasets) on untrusted external frontier models, the Control Plane can implement a local mathematical transformation layer. This utilizes AlphaTensor's **Change of Basis** technique, which plays games in randomly sampled unimodular bases to guarantee that exact factorizations discovered in an abstracted space remain mathematically correct when mapped back to the canonical basis. By applying an invertible change of basis locally, external models compute mathematically exact results on the randomized data, which SaeGyeol decodes locally using the inverse matrix—guaranteeing that raw parameters are never exposed to external services.
- **Empirical Hardware-Tailored Routing and Local Optimization**: To optimize the physical execution of workloads across hybrid local GPUs, sovereign private clouds, and external APIs, SaeGyeol can train its routing agent using an empirical, multi-objective reinforcement learning reward. This builds directly on AlphaTensor's hardware-tailored optimization pipeline, where feeding the **measured physical runtime of the compiled algorithm** on V100 GPUs and TPU v2 devices back to the agent allowed it to discover denser, compiler-friendly factorizations tailored directly to physical compiler fusion rules with zero prior hardware knowledge. By continuously monitoring the real-world execution times and compiler stacks of local on-premise hardware, the Control Plane can autonomously discover optimized routing profiles tailored to the institution's specific localized silicon configurations.