DeepMind

Discovering faster matrix multiplication algorithms with reinforcement learning

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

Example of a fast matrix multiplication algorithm (Strassen '69)

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix}$$

$$\begin{aligned}
p_1 &= (x_{11} + x_{22})(y_{11} + y_{22}), \\
p_2 &= (x_{11} + x_{22})y_{11}, \\
p_3 &= x_{11}(y_{12} - y_{22}), \\
p_4 &= x_{22}(-y_{11} + y_{12}), \\
p_5 &= (x_{11} + x_{12})y_{22}, \\
p_6 &= (-x_{11} + x_{21})(y_{11} + y_{12}), \\
p_7 &= (x_{12} - x_{22})(y_{21} + y_{22}).
\end{aligned}$$

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{pmatrix}.$$

$$\left(\begin{array}{c|c} \cdot & \cdot \\ \hline \cdot & \cdot \end{array}\right) \cdot \left(\begin{array}{c|c} \cdot & \cdot \\ \hline \cdot & \cdot \end{array}\right) = \left(\begin{array}{c|c} \cdot & \cdot \\ \hline \cdot & \cdot \end{array}\right)$$

$$\rightarrow C(n) \leq 7C(n/2) + O(n^2), \quad C(1) = 1$$

Theorem (Strassen)

We can multiply $n \times n$ matrices with $O(n^{\log_2(7)}) = O(n^{2.81})$ arithmetic operations.

Main idea

Fix matrix size (e.g. 2x2), use Reinforcement Learning to generate this algorithm line by line

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix}$$

$$\begin{aligned} p_1 &= (x_{11} + x_{22})(y_{11} + y_{22}), \\ p_2 &= (x_{11} + x_{22})y_{11}, \\ p_3 &= x_{11}(y_{12} - y_{22}), \\ p_4 &= x_{22}(-y_{11} + y_{12}), \\ p_5 &= (x_{11} + x_{12})y_{22}, \\ p_6 &= (-x_{11} + x_{21})(y_{11} + y_{12}), \\ p_7 &= (x_{12} - x_{22})(y_{21} + y_{22}). \end{aligned}$$

At the end of every episode, give non-zero reward if the algorithm is incorrect (we can check symbolically)

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{pmatrix}.$$

RL TLDR: do random things many times, choose best trajectories (according to reward), reinforce (learn to do more of those actions in similar situations)

Matrix Multiplication Tensor

Lets learn to find many different algorithms, from simple to complex, providing a curriculum for the agent

Matrix Multiplication Tensor

Lets learn to find many different algorithms, from simple to complex, providing a curriculum for the agent

Matrix multiplication is a bilinear operation.

Lets find algorithms for many bilinear operations.

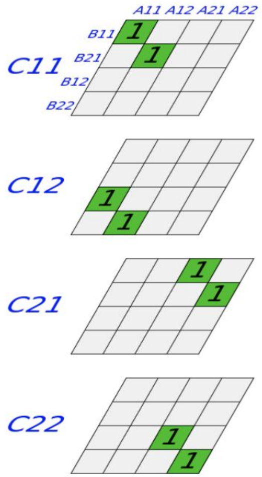
Matrix Multiplication Tensor

The matrix multiplication operator can be represented by a tensor.

The operator representing the multiplication of two NxN matrices, is a tensor of dimension $N^2 x N^2 x N^2$.

For 2x2 Matrix multiplications:

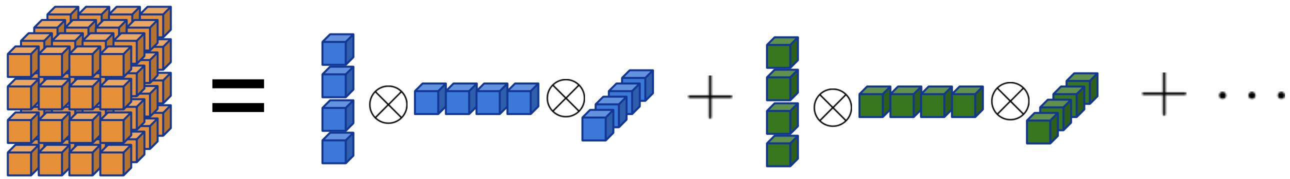
$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$



Matrix Multiplication Algorithm

Decompose this tensor into a factor decompositions.

$$T = \sum_{q \leq R} u_q \otimes v_q \otimes w_q$$



T $\quad$ $u_1 \otimes v_1 \otimes w_1$ $\quad$ $u_2 \otimes v_2 \otimes w_2$

Matrix Multiplication Algorithm

Decompose this tensor (cube) into a factor (vectors) decompositions.

$$T = \sum_{q \leq R} u_q \otimes v_q \otimes w_q$$

$$C = AB = \sum_{q=1}^{R} \langle u_q, A \rangle \langle v_q, B \rangle w_q$$

Matrix Multiplication Algorithm

Decompose this tensor (cube) into a factor (vectors) decompositions.

$$T = \sum_{q \leq R} u_q \otimes v_q \otimes w_q$$

$$C = AB = \sum_{q=1}^{\boxed{R}} \langle u_q, A \rangle \langle v_q, B \rangle w_q$$

Matrix Multiplication Algorithm

Decompose this tensor (cube) into a factor (vectors) decompositions.

$$T = \sum_{q \le R} u_q \otimes v_q \otimes w_q$$

$$C = AB = \sum_{q=1}^{\boxed{R}} \langle u_q, A \rangle \langle v_q, B \rangle w_q$$

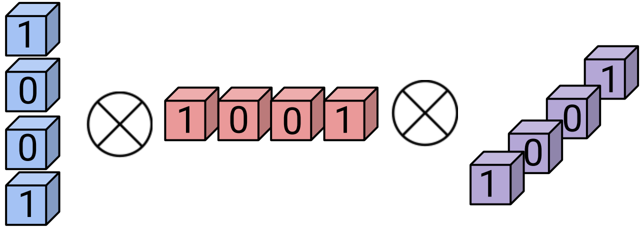
Can multiply matrices of arbitrary size through recursion

Strassen's algorithm

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix}$$

$$\begin{aligned}
p_1 &= (x_{11} + x_{22})(y_{11} + y_{22}), \\
p_2 &= (x_{11} + x_{22})y_{11}, \\
p_3 &= x_{11}(y_{12} - y_{22}), \\
p_4 &= x_{22}(-y_{11} + y_{12}), \\
p_5 &= (x_{11} + x_{12})y_{22}, \\
p_6 &= (-x_{11} + x_{21})(y_{11} + y_{12}), \\
p_7 &= (x_{12} - x_{22})(y_{21} + y_{22}).
\end{aligned}$$

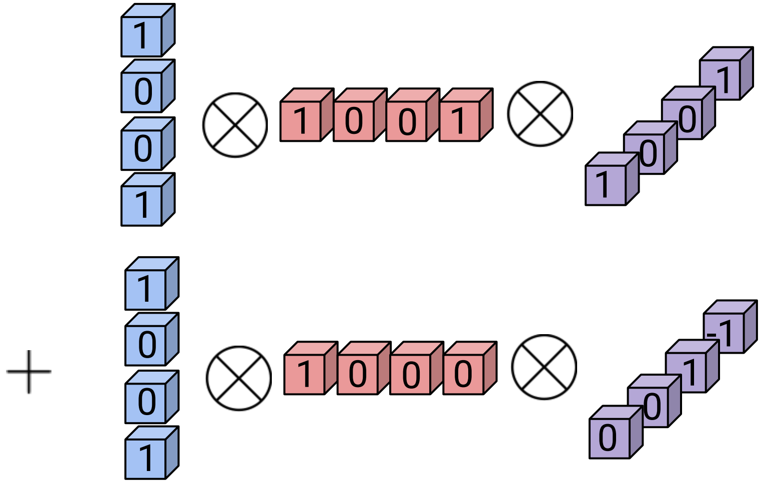
$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{pmatrix}.$$



Strassen's algorithm

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix}$$

$$\begin{aligned}
p_1 &= (x_{11} + x_{22})(y_{11} + y_{22}), \\
p_2 &= (x_{11} + x_{22})y_{11}, \\
p_3 &= x_{11}(y_{12} - y_{22}), \\
p_4 &= x_{22}(-y_{11} + y_{12}), \\
p_5 &= (x_{11} + x_{12})y_{22}, \\
p_6 &= (-x_{11} + x_{21})(y_{11} + y_{12}), \\
p_7 &= (x_{12} - x_{22})(y_{21} + y_{22}).
\end{aligned}$$



$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{pmatrix}.$$

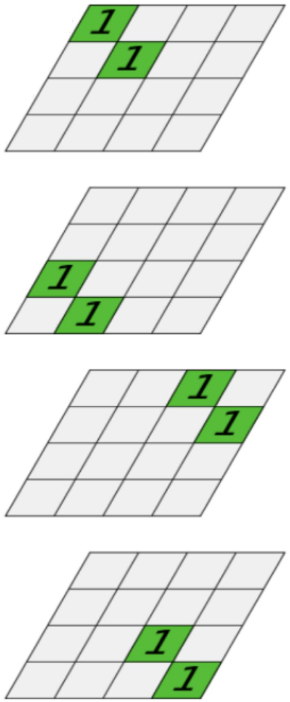
Strassen's algorithm

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \cdot \begin{pmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{pmatrix}$$

$$\begin{aligned}
p_1 &= (x_{11} + x_{22})(y_{11} + y_{22}), \\
p_2 &= (x_{11} + x_{22})y_{11}, \\
p_3 &= x_{11}(y_{12} - y_{22}), \\
p_4 &= x_{22}(-y_{11} + y_{12}), \\
p_5 &= (x_{11} + x_{12})y_{22}, \\
p_6 &= (-x_{11} + x_{21})(y_{11} + y_{12}), \\
p_7 &= (x_{12} - x_{22})(y_{21} + y_{22}).
\end{aligned}$$

$$\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix} = \begin{pmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{pmatrix}.$$

Rank-7 factorization



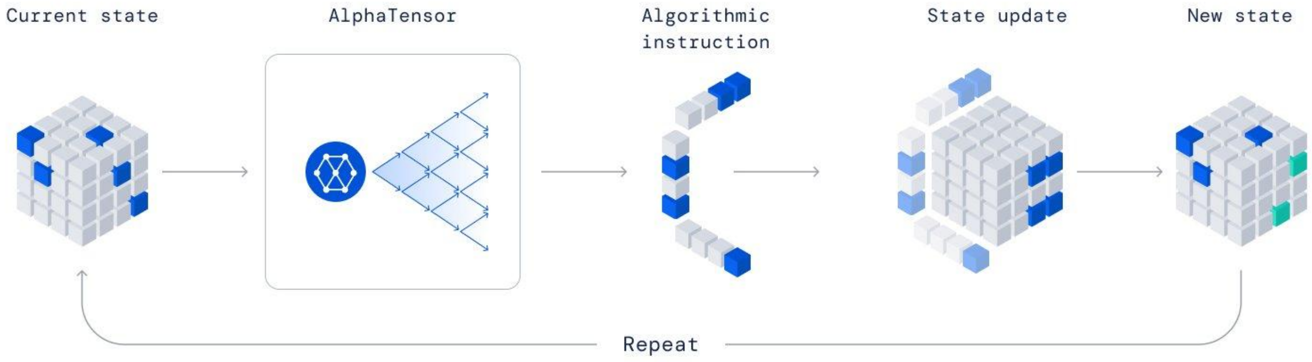
Modeling as a ML problem

Maths problem

Find low-rank decompositions of the matrix multiplication tensor

Modeling

Find shortest path to all-zero tensor



Modeling as a ML problem

Maths problem

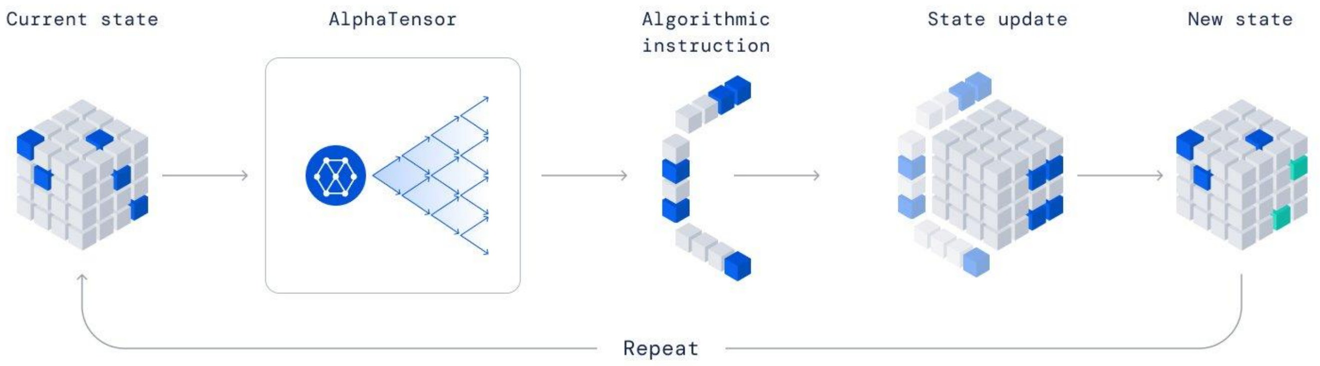
Find low-rank decompositions of the matrix multiplication tensor

Modeling

Find shortest path to all-zero tensor

Difficulties:

- Only one tensor to decompose
- No training data
- Huge action space
- Symmetries (e.g., permutation invariance)



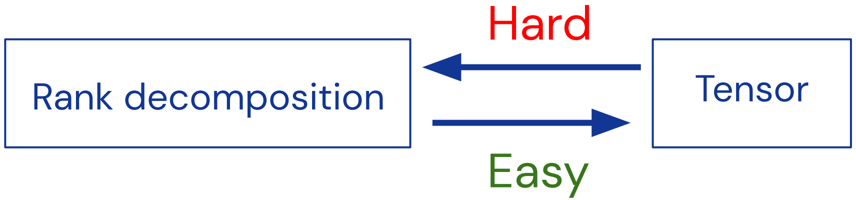
Ingredient #1: Synthetic data

- We generally require **lots of data** to train powerful ML models.
- In maths, abundant data is rarely available $\Rightarrow$ rely instead on synthetic data.

Ingredient #1: Synthetic data

- We generally require **lots of data** to train powerful ML models.
- In maths, abundant data is rarely available ⇒ rely instead on synthetic data.

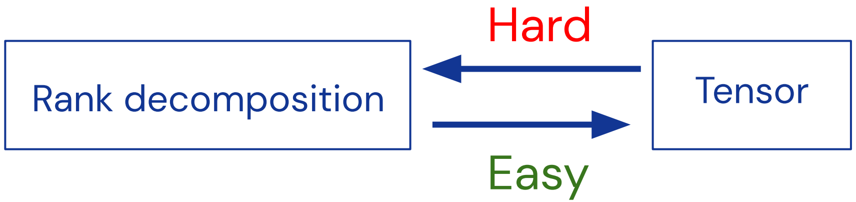
Generate (tensor, factorization) pairs, by generating *random* factorizations.



Ingredient #1: Synthetic data

- We generally require **lots of data** to train powerful ML models.
- In maths, abundant data is rarely available ⇒ rely instead on synthetic data.

Generate (tensor, factorization) pairs, by generating *random* factorizations.



- Training the network on data coming from actors in addition to synthetic data

Potential difficulty: The distribution of synthetic data can be far from that of the target.

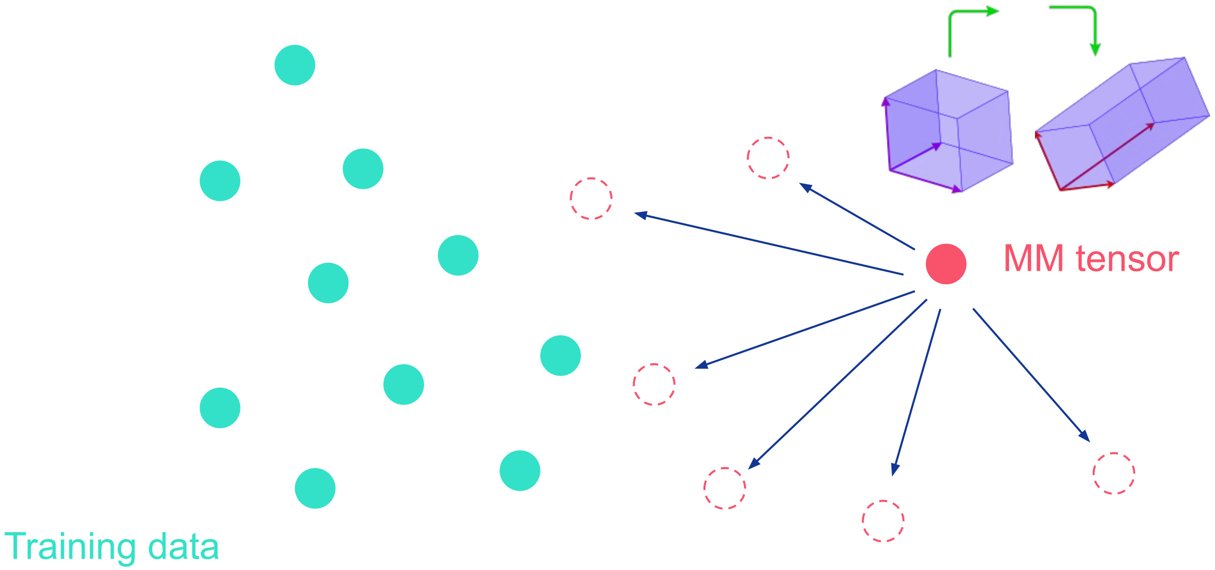
Ingredient #2: Diversifying the target

- In ML, we generally care about the **average** performance across many datapoints (test data).
- In many maths problems, we care about predicting the right result on **one** target (e.g., one tensor / one theorem …)

Ingredient #2: Diversifying the target

- In ML, we generally care about the **average** performance across many datapoints (test data).
- In many maths problems, we care about predicting the right result on **one** target (e.g., one tensor / one theorem ...)

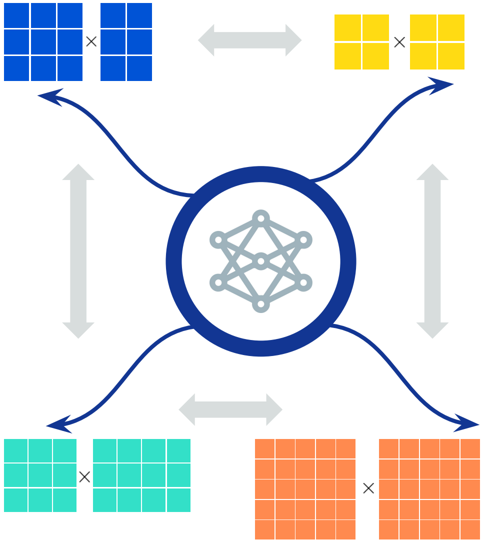
Express the target in mathematically equivalent ways – change the basis.



Ingredient #3: Train a generalist agent rather than several experts

$$
\begin{aligned}
h_1 &= a_6\,(b_1 - b_7 + b_9)\\
h_2 &= (a_1 + a_4)\,(b_1 - b_2 + b_5)\\
h_3 &= a_5\,(-b_3 - b_4 + b_6)\\
h_4 &= -\,(a_3 + a_6)\,(b_4 + b_7 - b_8)\\
h_5 &= b_8\,(a_1 + a_3)\\
h_6 &= b_3\,(a_4 + a_5)\\
h_7 &= a_2\,(b_5 + b_6 + b_9)\\
h_8 &= -b_5\,(a_1 + a_2 + a_4 + a_5)\\
h_9 &= (b_3 + b_5)\,(a_1 + a_4 + a_5)\\
h_{10} &= a_1\,(b_2 + b_3 - b_8)\\
h_{11} &= (a_1 - a_6)\,(b_1 - b_8)\\
h_{12} &= b_9\,(a_2 - a_3)\\
h_{13} &= b_4\,(a_2 - a_3 + a_5 - a_6)\\
h_{14} &= (b_4 + b_9)\,(-a_2 + a_3 + a_6)\\
h_{15} &= b_1\,(a_4 + a_6)\\
y_1 &= h_1 + h_{11} - h_{12} - h_{14} - h_4 + h_5\\
y_2 &= -h_1 + h_{12} + h_{13} + h_{14} + h_{15}\\
y_3 &= h_{10} + h_5 + h_6 - h_8 - h_9\\
y_4 &= -h_{10} + h_{11} + h_{15} - h_2 - h_6 + h_9\\
y_5 &= -h_{12} - h_6 + h_7 + h_8 + h_9\\
y_6 &= h_{12} + h_{13} + h_{14} + h_3 + h_6
\end{aligned}
$$

$$
\begin{aligned}
h_1 &= (a_1 + a_4)\,(b_1 + b_4)\\
h_2 &= a_1\,(b_2 - b_4)\\
h_3 &= (a_2 - a_4)\,(b_3 + b_4)\\
h_4 &= b_1\,(a_3 + a_4)\\
h_5 &= a_4\,(-b_1 + b_3)\\
h_6 &= -\,(a_1 - a_3)\,(b_1 + b_2)\\
h_7 &= b_4\,(a_1 + a_2)\\
y_1 &= h_1 + h_3 + h_5 - h_7\\
y_2 &= h_4 + h_5\\
y_3 &= h_2 + h_7\\
y_4 &= h_1 + h_2 - h_4 + h_6
\end{aligned}
$$



$$
\begin{aligned}
h_1 &= -\,(a_4 + a_5 + a_8)\,(b_2 - b_3 + b_4 - b_8)\\
h_2 &= (a_1 + a_2 + a_7)\,(b_1 - b_5 - b_6 + b_7)\\
h_3 &= (b_{12} - b_2)\,(-a_1 + a_3 + a_4)\\
h_4 &= -\,(b_6 - b_9)\,(a_2 - a_5 + a_6)\\
h_5 &= -\,(b_{11} + b_2)\,(a_1 - a_3 + a_7)\\
h_6 &= (b_{11} + b_6)\,(a_5 - a_6 + a_8)\\
h_7 &= b_{11}\,(-a_1 + a_3 - a_5 + a_6 - a_7 - a_8 + a_9)\\
h_8 &= b_2\,(-a_1 + a_3)\\
h_9 &= b_6\,(-a_5 + a_6)\\
h_{10} &= (a_5 + a_8)\,(b_{11} + b_2 - b_3 + b_4 + b_7 - b_8)\\
h_{11} &= (b_5 + b_6 - b_7)\,(a_1 + a_2 + a_7 + a_8)\\
h_{12} &= (b_2 - b_3 + b_4)\,(a_4 + a_5 + a_7 + a_8)
\end{aligned}
$$

$$\vdots$$

$$
\begin{aligned}
y_1 &= h_{14} + h_{18} + h_{23} + h_4 + h_9\\
y_2 &= h_{14} + h_{16} + h_{20} + h_{21} + h_4 + h_9\\
y_3 &= h_{11} + h_2 + h_{22} - h_{23} + h_{28}\\
y_4 &= -h_{18} - h_{25} - h_4 - h_8 - h_9\\
y_5 &= h_{17} + h_{26} - h_3 - h_8 - h_9\\
y_6 &= h_{27} - h_5 + h_6 + h_7 + h_8 + h_9\\
y_7 &= h_{13} + h_{14} - h_{19} + h_2 + h_5 - h_8
\end{aligned}
$$

$$
\begin{aligned}
h_1 &= -a_{16}b_4\\
h_2 &= a_1\,(-b_{23} + b_3)\\
h_3 &= (a_1 + a_5)\,(b_1 + b_2 + b_{23} - b_5)\\
h_4 &= (a_1 - a_{20} + a_5)\,(-b_{23} + b_3 + b_4)\\
h_5 &= (a_{20} + a_3 - a_5)\,(b_{14} - b_2 + b_{22})\\
h_6 &= (a_3 - a_5)\,(b_{11} + b_{15} + b_2 - b_{22})\\
h_7 &= -\,(b_{14} + b_{17} - b_7)\,(a_{18} - a_{20} - a_3 + a_5)\\
h_8 &= b_5\,(-a_{16} + a_{17} - a_{20})\\
h_9 &= (-a_{20} + a_4 + a_5)\,(-b_1 + b_{11} - b_{17} + b_{19} + b_{20} + b_{21})\\
h_{10} &= (a_2 - a_{20} + a_5)\,(b_{15} + b_{25} - b_5 + b_6 - b_7 + b_9)\\
h_{11} &= a_{18}\,(b_{12} - b_{14} + b_2)\\
h_{12} &= (a_1 - a_{16} - a_{20} + a_5)\,(b_{13} + b_{18} + b_3 + b_4 + b_8)
\end{aligned}
$$

$$\vdots$$

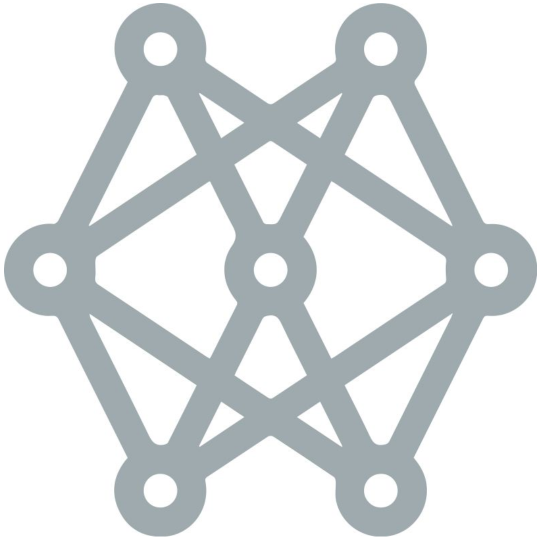
$$
\begin{aligned}
y_1 &= h_{15} + h_{18} - h_{19} - h_{21} - h_{22} - h_{24} + h_{25} - h_{27} - h_{28} + h_{31} - h_{32}\\
y_2 &= -h_{46} + h_{71} - h_{73} - h_{78} + h_{80} - h_{82} + h_{83} + h_{84} + h_{87} - h_{91} + h_{92}\\
y_3 &= -h_{46} - h_{49} + h_{53} + h_{63} + h_{72} + h_{80} + h_{84} + h_{94}\\
y_4 &= -h_{16} + h_{25} - h_{27} - h_{28} - h_{32} + h_{33} - h_9\\
y_5 &= h_{43} - h_{46} - h_{54} - h_{56} + h_{70} + h_{72} + h_{80} + h_{84} + h_{87} - h_{92}\\
y_6 &= h_{13} - h_{18} - h_{20} + h_{21} + h_{29} + h_{32} + h_{34} - h_{35} + h_{38} - h_{40} + h_6\\
y_7 &= h_{44} - h_{45} - h_{46} - h_{54} + h_{55} - h_{60} - h_{66} + h_{89} + h_{95} + h_{97}\\
y_8 &= h_{41} - h_{42} + h_{43} - h_{52} + h_{59} + h_{61} + h_{84} + h_{88} + h_{89} + h_{98}\\
y_9 &= h_{11} + h_{13} - h_{18} + h_{29} + h_{32} - h_{40} + h_5 + h_6 - h_7\\
y_{10} &= h_{41} - h_{42} + h_{43} + h_{44} - h_{45} - h_{46} - h_{52} - h_{54} - h_{56} + h_{61} + h_{64} - h_{66} + h_{84}
\end{aligned}
$$

Generalist agent vs expert

- **Better performance through transfer:** Generalist agent getting better results (more efficient algorithms) compared to individual “experts”.

- **More efficient:** Can generate efficient algorithms tailored for each matrix size (with just one experiment!)

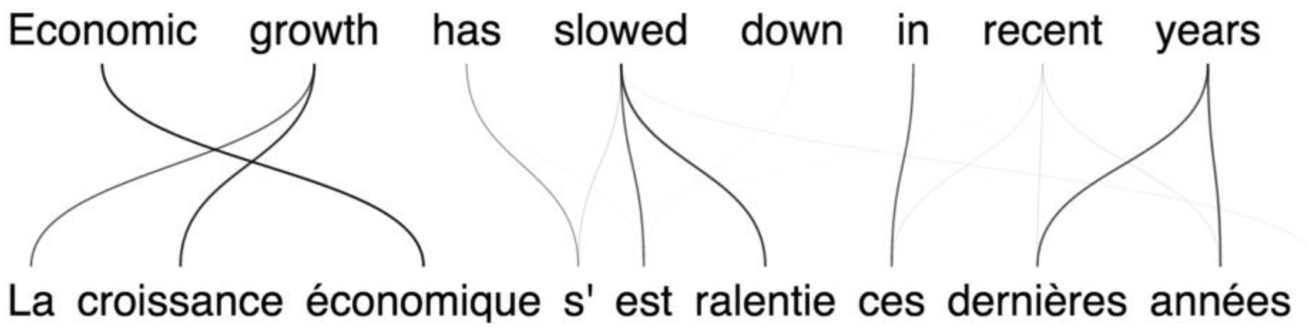
Ingredient #4: Training large and deep models adapted to the task



Beyond shallow fully-connected layers

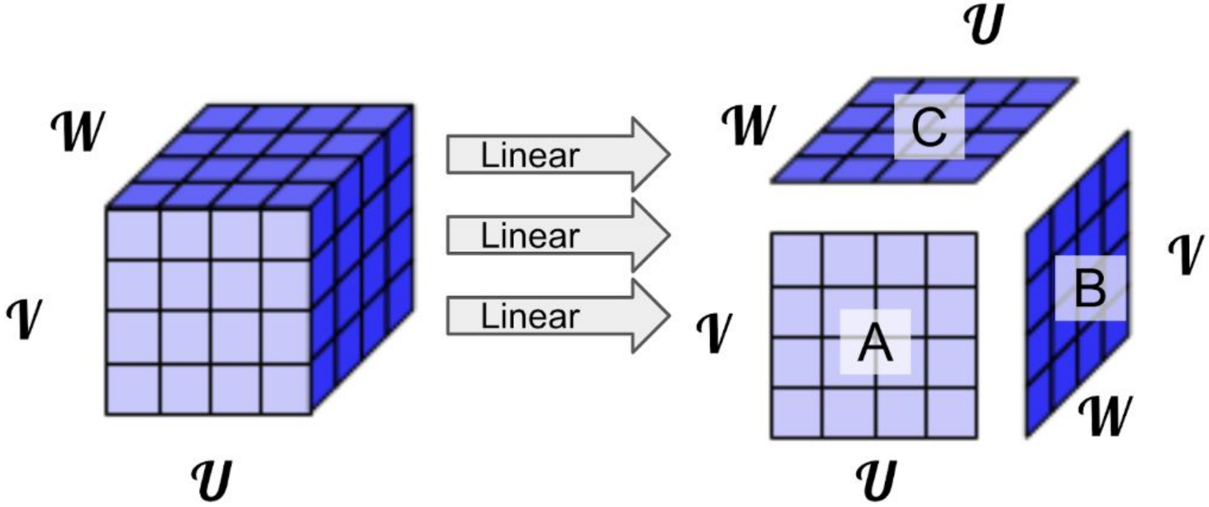
Ingredient #4: Training large and deep models adapted to the task: attention

Attention and transformers have now become ubiquitous in ML models



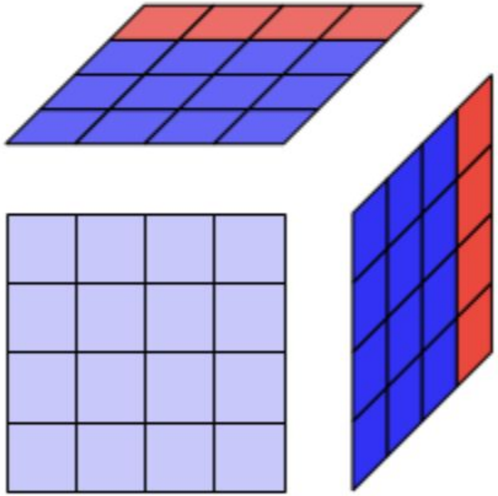
Ingredient #4: Training large and deep models adapted to the task: attention

Adapt the architecture to the task at hand by incorporating symmetries and prior knowledge about the problem

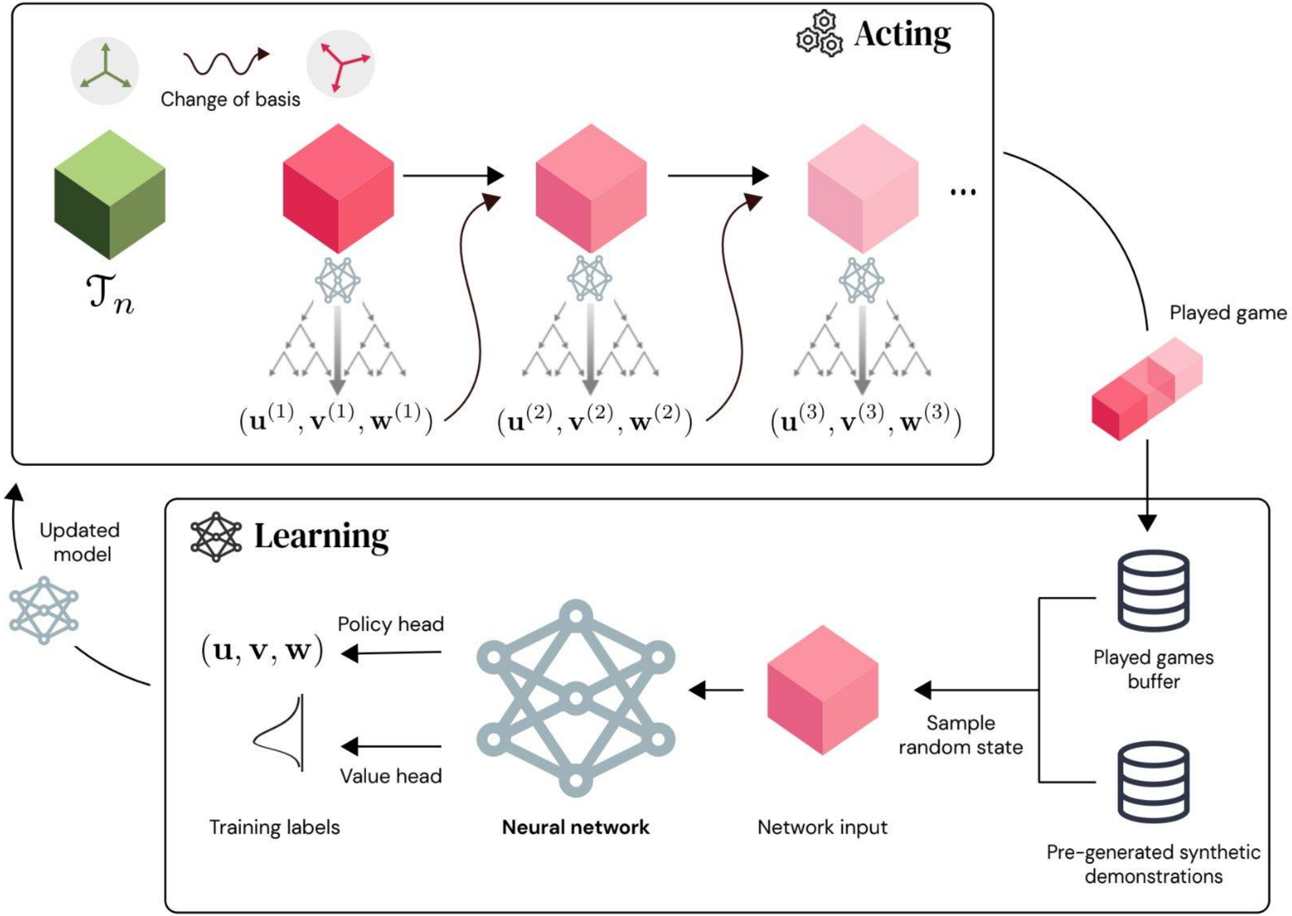


Ingredient #4: Training large and deep models adapted to the task: attention

Adapt the architecture to the task at hand by incorporating symmetries and prior knowledge about the problem



Overall system



Results on matrix multiplication tensors

Size (n, m, p)	Best method known	Best rank known	*AlphaTensor* rank Modular	Standard
$(2,2,2)$	(Strassen, 1969)	7	7	7
$(3,3,3)$	(Laderman, 1976)	23	23	23
$(4,4,4)$	(Strassen, 1969) $(2,2,2) \otimes (2,2,2)$	49	**47**	49
$(5,5,5)$	$(3,5,5) + (2,5,5)$	98	**96**	98
$(2,2,3)$	$(2,2,2) + (2,2,1)$	11	11	11
$(2,2,4)$	$(2,2,2) + (2,2,2)$	14	14	14
$(2,2,5)$	$(2,2,2) + (2,2,3)$	18	18	18
$(2,3,3)$	(Hopcroft and Kerr, 1971)	15	15	15
$(2,3,4)$	(Hopcroft and Kerr, 1971)	20	20	20
$(2,3,5)$	(Hopcroft and Kerr, 1971)	25	25	25
$(2,4,4)$	(Hopcroft and Kerr, 1971)	26	26	26
$(2,4,5)$	(Hopcroft and Kerr, 1971)	33	33	33
$(2,5,5)$	(Hopcroft and Kerr, 1971)	40	40	40
$(3,3,4)$	(Smirnov, 2013)	29	29	29
$(3,3,5)$	(Smirnov, 2013)	36	36	36
$(3,4,4)$	(Smirnov, 2013)	38	38	38
$(3,4,5)$	(Smirnov, 2013)	48	**47**	**47**
$(3,5,5)$	(Sedoglavic and Smirnov, 2021)	58	58	58
$(4,4,5)$	$(4,4,2) + (4,4,3)$	64	**63**	**63**
$(4,5,5)$	$(2,5,5) \otimes (2,1,1)$	80	**76**	**76**

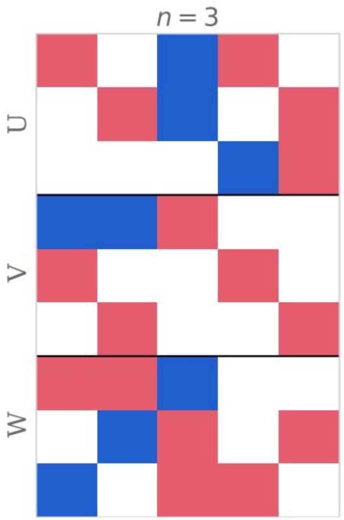
Beyond standard matrix multiplication

Our procedure can be applied to find algorithms for arbitrary bilinear maps

Beyond standard matrix multiplication

Our procedure can be applied to find algorithms for arbitrary bilinear maps

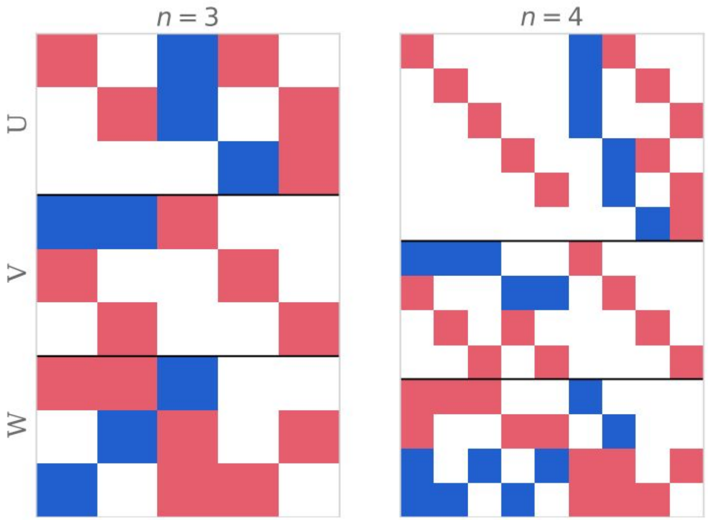
Example: skew-symmetric matrix-vector product



Beyond standard matrix multiplication

Our procedure can be applied to find algorithms for arbitrary bilinear maps

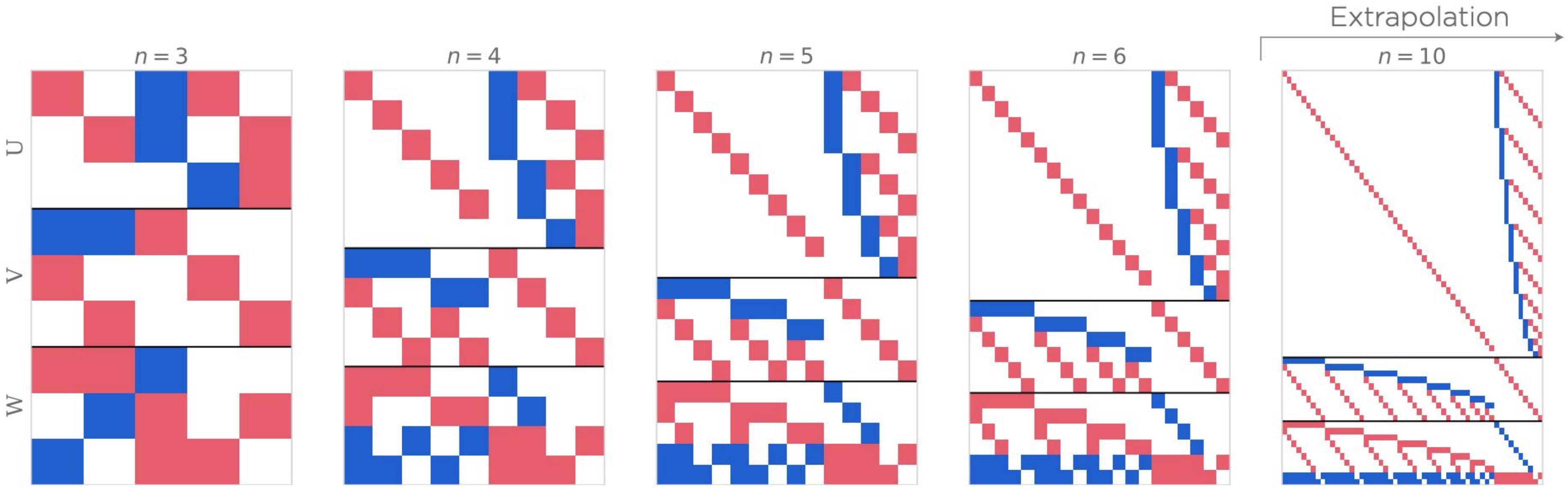
Example: skew-symmetric matrix-vector product



Beyond standard matrix multiplication

Our procedure can be applied to find algorithms for arbitrary bilinear maps

Example: skew-symmetric matrix-vector product



Beyond standard matrix multiplication

Our procedure can be applied to find algorithms for arbitrary bilinear maps

Example: skew-symmetric matrix-vector product

Input: $n \times n$ skew-symmetric matrix $\mathbf{A}$, vector $\mathbf{b}$.
Output: The resulting vector $\mathbf{c} = \mathbf{A}\mathbf{b}$ computed in $\frac{(n-1)(n+2)}{2}$ multiplications.

for $i = 1, \dots, n-2$ **do**
  for $j = i+1, \dots, n$ **do**
    $w_{ij} = a_{ij}(b_j - b_i)$ ▷ Computing the first $(n-2)(n+1)/2$ intermediate products
for $i = 1, \dots, n$ **do**
  $q_i = b_i \sum_{j=1}^{n} a_{ji}$ ▷ Computing the final n intermediate products
for $i = 1, \dots, n-2$ **do**
  $c_i = \sum_{j=1}^{i-1} w_{ji} + \sum_{j=i+1}^{n} w_{ij} - q_i$
$c_{n-1} = -\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-2} w_{ij} - \sum_{j=1}^{n-2} w_{jn} + \sum_{i=1, i \neq n-1}^{n} q_i$
$c_n = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} + \sum_{i=1}^{n-1} q_i$

Beyond standard matrix multiplication

Our procedure can be applied to find algorithms for arbitrary bilinear maps

Example: skew-symmetric matrix-vector product

Input: $n \times n$ skew-symmetric matrix $\mathbf{A}$, vector $\mathbf{b}$.

Output: The resulting vector $\mathbf{c} = \mathbf{A}\mathbf{b}$ computed in $\boxed{\frac{(n-1)(n+2)}{2} \text{ multiplications.}}$

Improves over previous best know result [Ye, Lim, 2018]

1: **for** $i = 1, \ldots, n-2$ **do**

2: **for** $j = i+1, \ldots, n$ **do**

3: $w_{ij} = a_{ij}(b_j - b_i)$ ▷ Computing the first $(n-2)(n+1)/2$ intermediate products

4: **for** $i = 1, \ldots, n$ **do**

5: $q_i = b_i \sum_{j=1}^{n} a_{ji}$ ▷ Computing the final n intermediate products

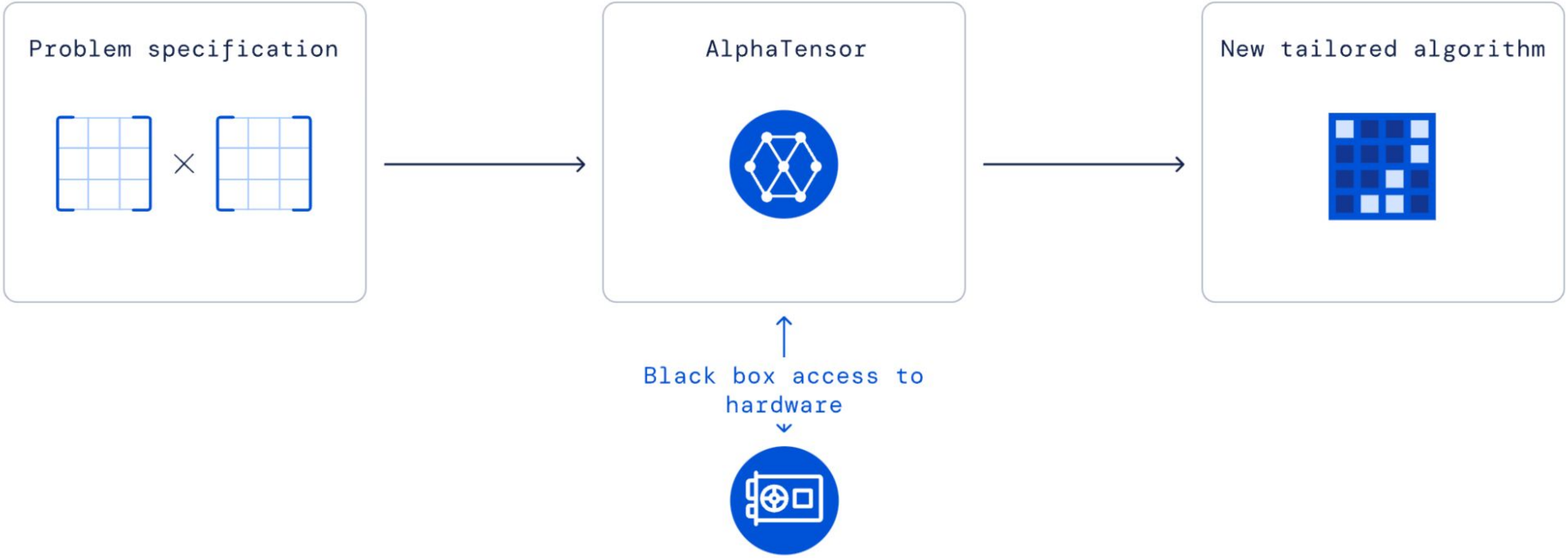
6: **for** $i = 1, \ldots, n-2$ **do**

7: $c_i = \sum_{j=1}^{i-1} w_{ji} + \sum_{j=i+1}^{n} w_{ij} - q_i$

8: $c_{n-1} = -\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-2} w_{ij} - \sum_{j=1}^{n-2} w_{jn} + \sum_{i=1, i \neq n-1}^{n} q_i$

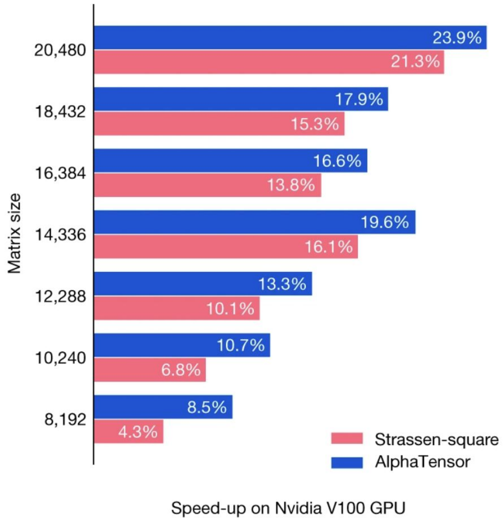
9: $c_n = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} + \sum_{i=1}^{n-1} q_i$

Beyond rank optimization



Runtime on real-world hardware

Reward proportional to the execution time of the algorithm



Algorithm tailored to the target hardware (e.g., algorithm optimized on CPU would not perform well on GPU)

Conclusions

- Transformed a maths/algorithmic problem into a game, and used 4 ingredients to make ML actually work

    1. If there is no data, generate synthetic data
    2. Diversifying the target
    3. Generalist agent, rather than expert
    4. Use large deep models, and embed prior knowledge into the architecture

- The resulting discovered algorithms outperform state-of-the-art algorithms in terms of rank

- Obtained system is very flexible and customizable (e.g., supporting finite fields, arbitrary tensors, reward, …)

Extra slides

Matrix Multiplication Algorithm

Decompose this tensor (cube) into a factor (vectors) decompositions.

$$T = \sum_{q \leq R} u_q \otimes v_q \otimes w_q$$

Algorithm 1: Meta-algorithm parameterized by $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$ for computing the matrix product $\mathbf{C} = \mathbf{AB}$. Note that R controls the number of multiplications between input matrix entries.

Parameters: $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$: length-$n^2$ vectors such that $\mathcal{T}_n = \sum_{r=1}^{R} \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$

Input: $\mathbf{A}, \mathbf{B}$: matrices of size $n \times n$

Output: $\mathbf{C} = \mathbf{AB}$

```
for r = 1, ..., R do
    m_r ← (u_1^(r) a_1 + ··· + u_{n^2}^(r) a_{n^2}) (v_1^(r) b_1 + ··· + v_{n^2}^(r) b_{n^2})
for i = 1, ..., n^2 do
    c_i ← w_i^(1) m_1 + ··· + w_i^(R) m_R
return C
```

for $r = 1, \ldots, R$ **do**

$\quad m_r \leftarrow \left(u_1^{(r)} a_1 + \cdots + u_{n^2}^{(r)} a_{n^2}\right)\left(v_1^{(r)} b_1 + \cdots + v_{n^2}^{(r)} b_{n^2}\right)$

for $i = 1, \ldots, n^2$ **do**

$\quad c_i \leftarrow w_i^{(1)} m_1 + \cdots + w_i^{(R)} m_R$

return $\mathbf{C}$

Matrix Multiplication Algorithm

Decompose this tensor (cube) into a factor (vectors) decompositions.

$$T = \sum_{q \le R} u_q \otimes v_q \otimes w_q$$

Algorithm 1: Meta-algorithm parameterized by $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$ for computing the matrix product $\mathbf{C} = \mathbf{AB}$. Note that R controls the number of multiplications between input matrix entries.

Parameters: $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$: length-$n^2$ vectors such that $\mathcal{T}_n = \sum_{r=1}^{R} \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$
Input: $\mathbf{A}, \mathbf{B}$: matrices of size $n \times n$
Output: $\mathbf{C} = \mathbf{AB}$

```
for r = 1, ..., R do
    m_r ← (u_1^(r) a_1 + ··· + u_{n^2}^(r) a_{n^2}) (v_1^(r) b_1 + ··· + v_{n^2}^(r) b_{n^2})
for i = 1, ..., n^2 do
    c_i ← w_i^(1) m_1 + ··· + w_i^(R) m_R
return C
```

$$m_r \leftarrow \left(u_1^{(r)} a_1 + \cdots + u_{n^2}^{(r)} a_{n^2}\right)\left(v_1^{(r)} b_1 + \cdots + v_{n^2}^{(r)} b_{n^2}\right)$$

$$c_i \leftarrow w_i^{(1)} m_1 + \cdots + w_i^{(R)} m_R$$