AlphaTensor

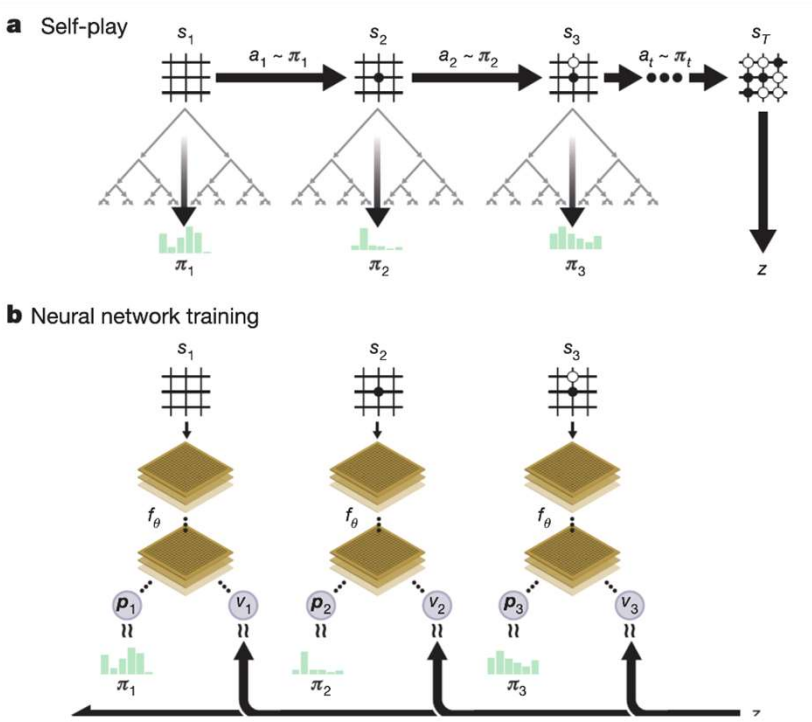
7 Nov 2022

John Tan Chong Min

Overview

- Neural Networks typically are implemented as Matrix (Tensor) multiplications
- A speedup in matrix multiplication (costly operation is multiplication step) will help to speed up Deep Learning
- AlphaTensor uses Deep Reinforcement Learning (DRL) to find an improvement to reduce multiplication steps in matrix multiplication
  - 47 for 4x4 by 4x4 (previously 49)
  - 96 for 5x5 by 5x5 (previously 98)

How AlphaGo Zero Works



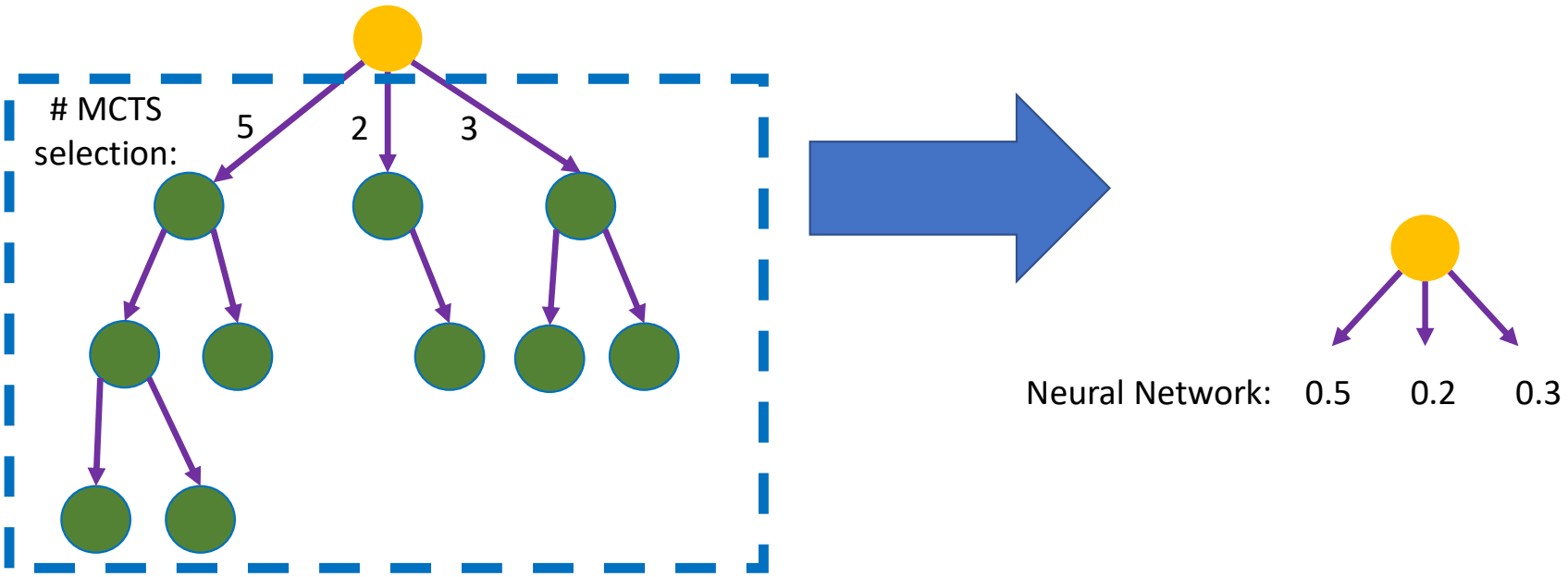
- Train so that policy network mimics the improved policy distribution π generated by MCTS
- Train so that value network output matches the game outcome for all time steps
- Joint Loss function:

$$(\boldsymbol{p}, v) = f_\theta(s) \text{ and } l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

- *z*: Actual game outcome, *v*: Predicted game outcome
- π : MCTS probability of next move, *p*: Predicted probability of next move
- c: L2 regularization constant (set at 10^{-4})

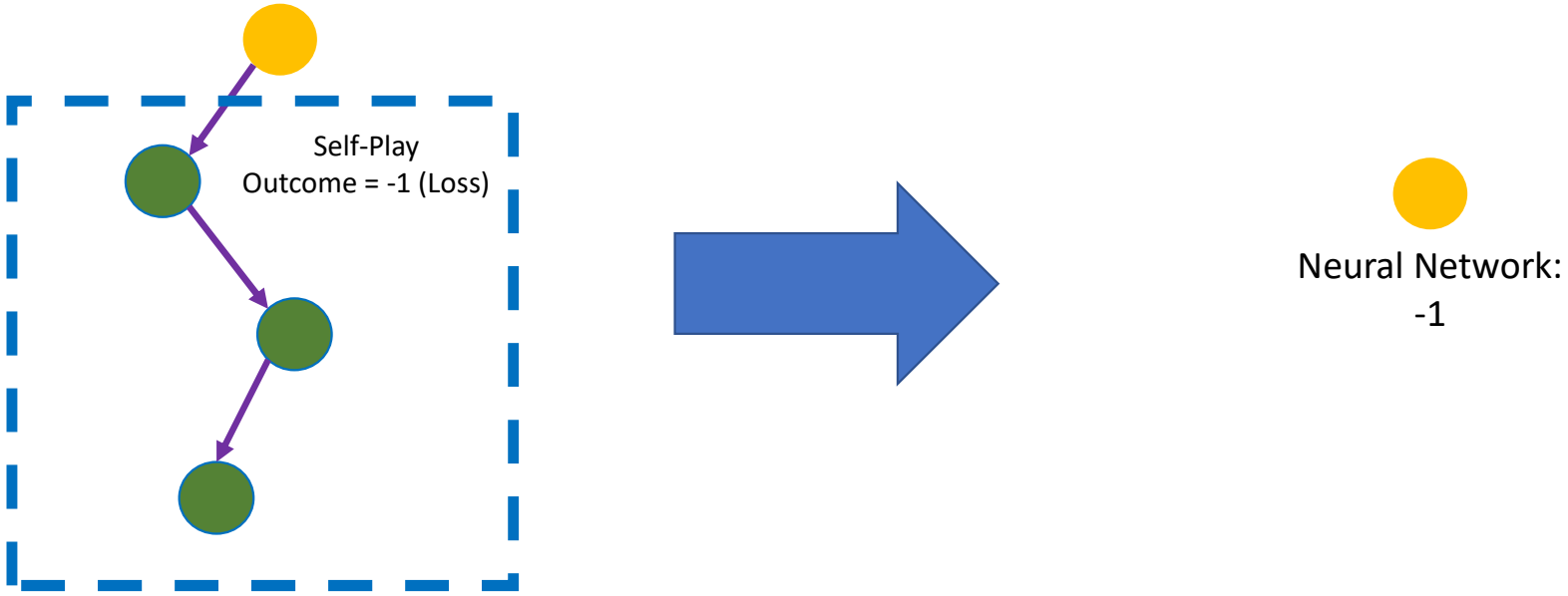
Policy Network (Breadth)

- Focus on moves with higher win rate!



Value Network (Depth)

- Look ahead without simulation!
- Even more accurate than Monte Carlo rollouts



AlphaTensor

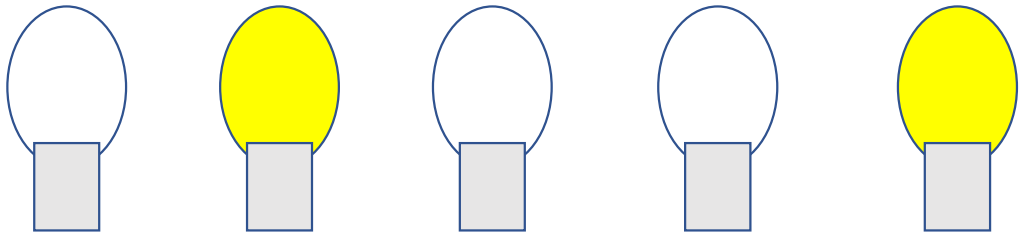
Single-player AlphaZero applied to Matrix Multiplication

Overview

- Formulate matrix multiplication as a single-player game, TensorGame
- At each step, player selects how to combine different entries of matrices to multiply
- Score assigned based on number of multiplications needed, and complexity of the resultant matrix

The Simplified Game

- You need to turn off all the lights of this row:



- Each turn, you can only turn on/off all lights at the same time of a few sets of bulbs
  - {1, 2}
  - {2, 4}
  - {3, 5}
  - {4, 5}

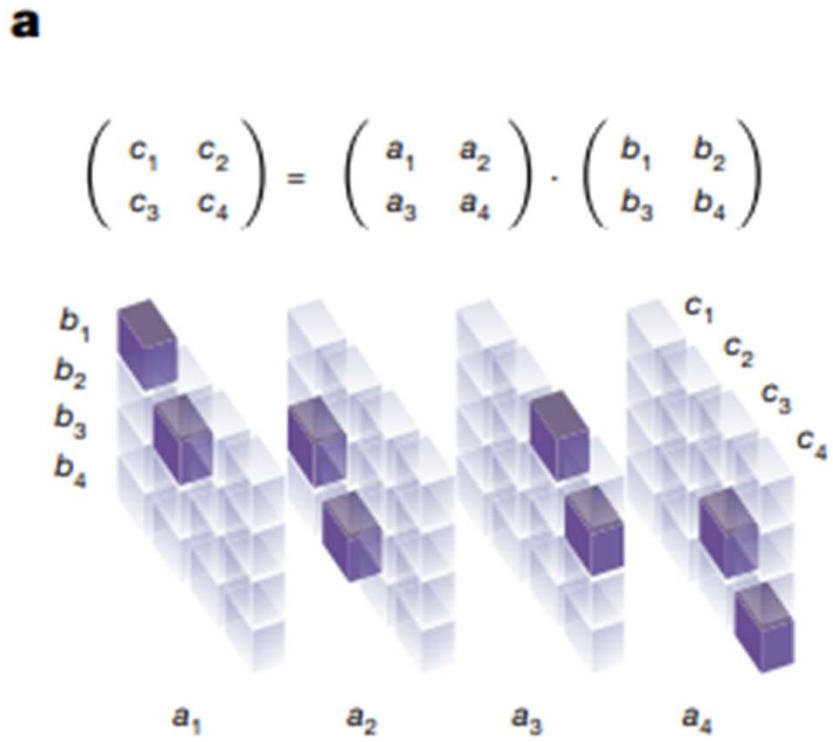
- How do you turn off all the lights with the least number of moves?

Normal approach to multiplication

- Need 8 product steps

- $c_1 = a_1b_1 + a_2b_3$
- $c_2 = a_1b_2 + a_2b_4$
- $c_3 = a_3b_1 + a_4b_3$
- $c_4 = a_3b_2 + a_4b_4$

Can be viewed as specifying a 3D 4x4x4 tensor (with entries being 0 and 1) to specify which are the product terms for resultant matrix *C*



Strassen's Algorithm

Assuming summation terms have 0 cost, we seek to reduce number of multiplication terms

- Form intermediate terms *m* based on (sum of selected *a* factors)(sum of selected *b* factors)
- Sum up selected *m* factors to form *c*
- We can achieve it in 7 multiplication steps!

b

$$m_1 = (a_1 + a_4)(b_1 + b_4)$$
$$m_2 = (a_3 + a_4)\,b_1$$
$$m_3 = a_1\,(b_2 - b_4)$$
$$m_4 = a_4\,(b_3 - b_1)$$
$$m_5 = (a_1 + a_2)\,b_4$$
$$m_6 = (a_3 - a_1)(b_1 + b_2)$$
$$m_7 = (a_2 - a_4)(b_3 + b_4)$$
$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

c

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Main algorithm

Algorithm 1

A meta-algorithm parameterized by $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$ for computing the matrix product $\mathbf{C}=\mathbf{AB}$. It is noted 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: $\mathbf{C}=\mathbf{AB}$

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

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

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

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

return $\mathbf{C}$

$$
\begin{aligned}
m_1 &= (a_1 + a_4)(b_1 + b_4) \\
m_2 &= (a_3 + a_4) b_1 \\
m_3 &= a_1 (b_2 - b_4) \\
m_4 &= a_4 (b_3 - b_1) \\
m_5 &= (a_1 + a_2) b_4 \\
m_6 &= (a_3 - a_1)(b_1 + b_2) \\
m_7 &= (a_2 - a_4)(b_3 + b_4) \\
c_1 &= m_1 + m_4 - m_5 + m_7 \\
c_2 &= m_3 + m_5 \\
c_3 &= m_2 + m_4 \\
c_4 &= m_1 - m_2 + m_3 + m_6
\end{aligned}
$$

$$
\mathbf{U} = \begin{pmatrix}
1 & 0 & 1 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & -1
\end{pmatrix}
$$

$$
\mathbf{V} = \begin{pmatrix}
1 & 1 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & -1 & 0 & 1 & 0 & 1
\end{pmatrix}
$$

$$
\mathbf{W} = \begin{pmatrix}
1 & 0 & 0 & 1 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}
$$

How to play the game

- Start with the desired 3D tensor for the matrix multiplication
- Each step *r*, we then subtract the tensor $\mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$
- Eventually after *R* steps, we will have subtracted $\mathcal{T}_n = \sum_{r=1}^{R} \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$
- If the list of steps are correct, we will get the **0** tensor

r = 1

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

- Definition of outer product (*u* is vector of length *m*, *v* vector of length *n*):

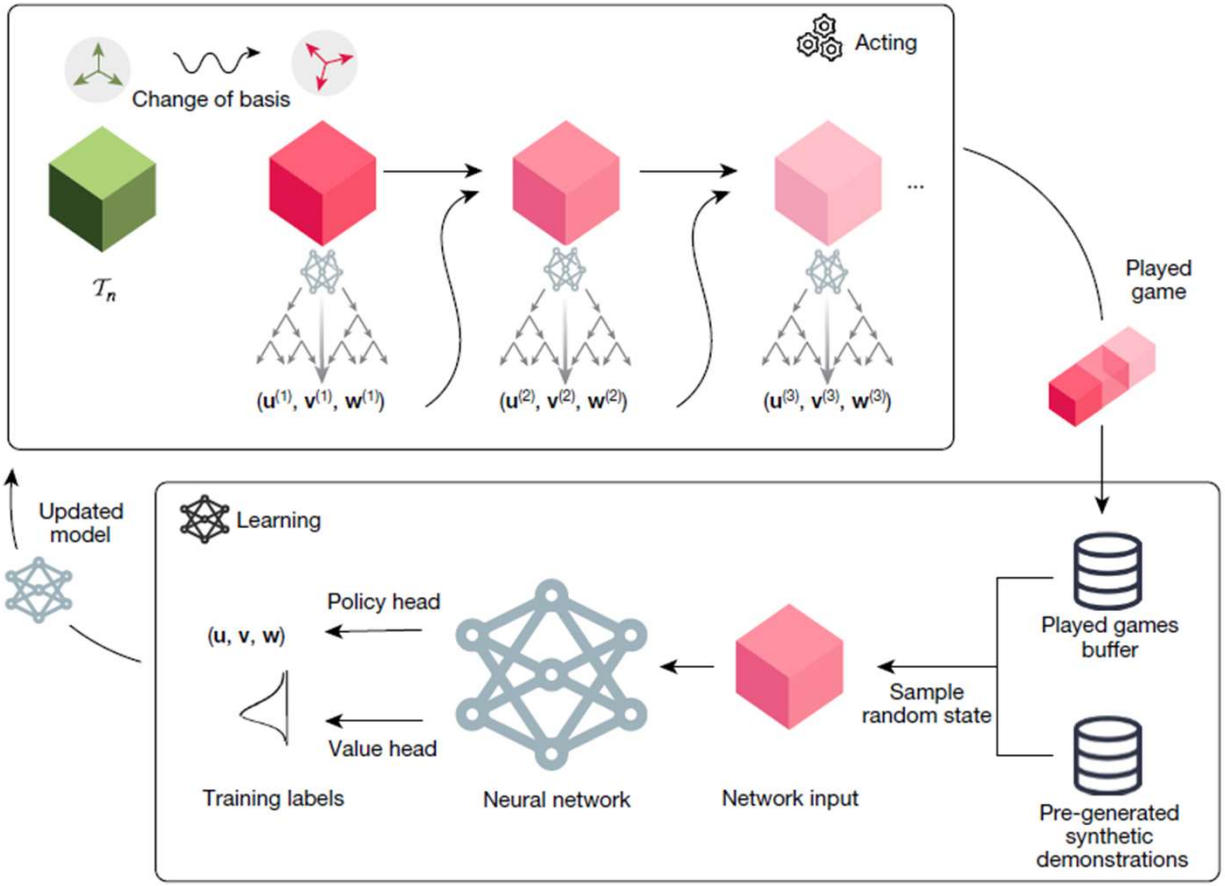
$$\mathbf{u} \otimes \mathbf{v} = \mathbf{A} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$

How to reward the agent

- Encourage finding shortest path: Each step is given -1 reward
- Encourage finding path closer to goal state: Terminal step given reward based on the complexity of the terminal tensor: $-\gamma(\mathcal{S}_{R_{\text{limit}}})$
- u, v, w constrained to be of the elements {-2, -1, 0, 1, 2}

Acting and Learning Process

- Use Monte Carlo Tree Search to sample state space
- Policy Network tells us how often to choose an action
- Value Network gives the value of the current state

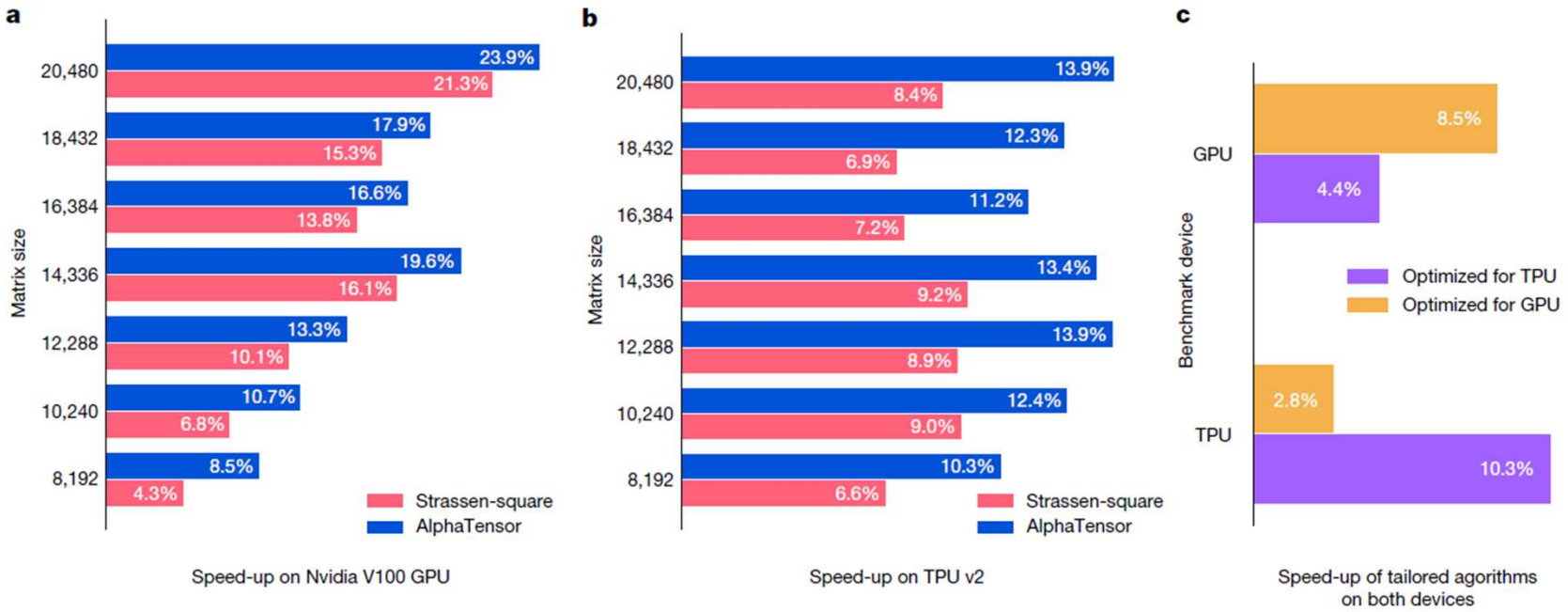


Additional Data Augmentation

- Synthetic demonstrations – Firstly, generate a custom goal tensor over R steps, then play the game using the known decomposition to give more “success” cases
  - Quite similar to Hindsight Experience Replay
- Change of Basis for Tensor Multiplication
  - Increase diversity of known situations
- Swapping a move with the last move (as game is order invariant)
  - Learning more from the same experience

Optimising for Hardware

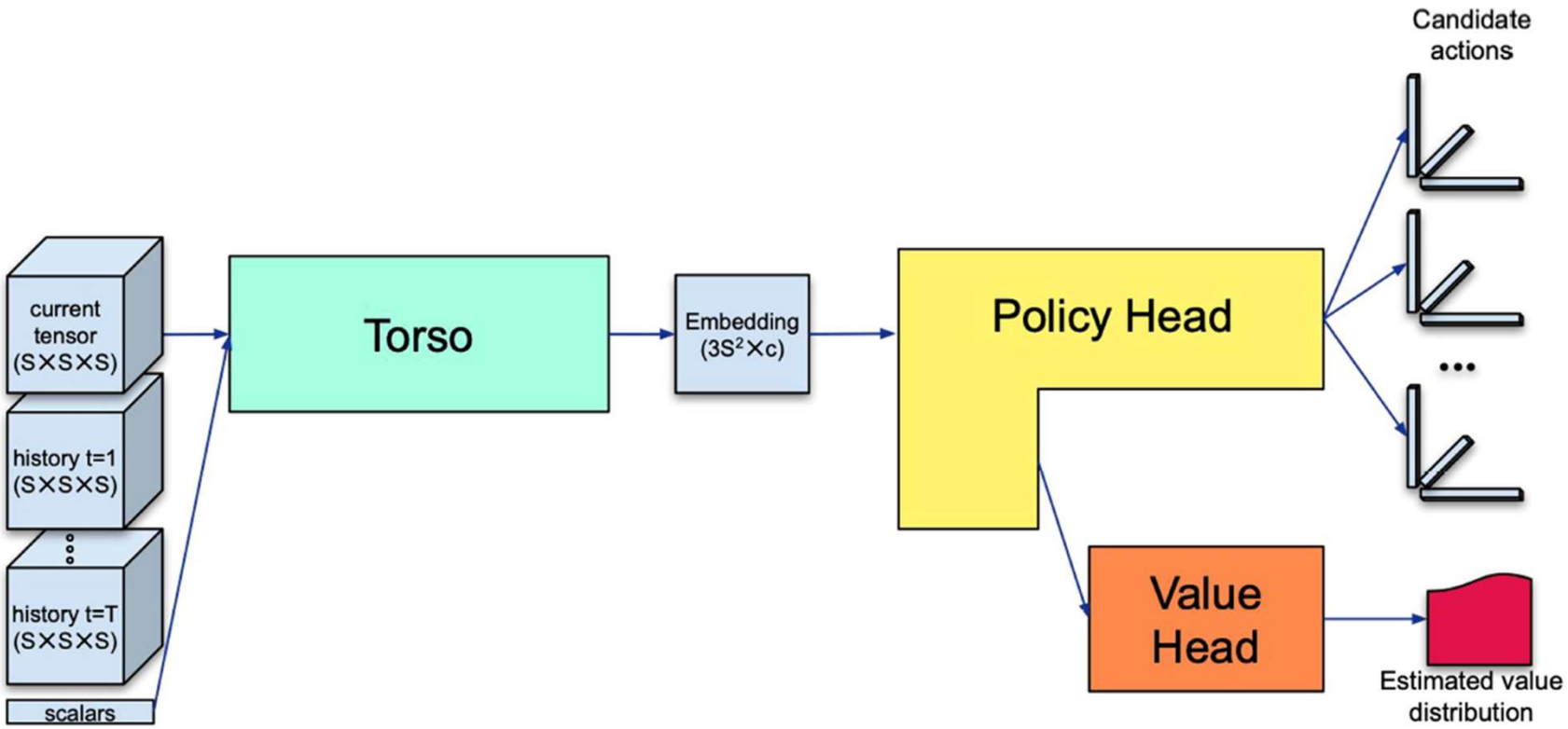
- Adding a runtime-based reward at terminal state
  - Longer the runtime, the more negative this reward



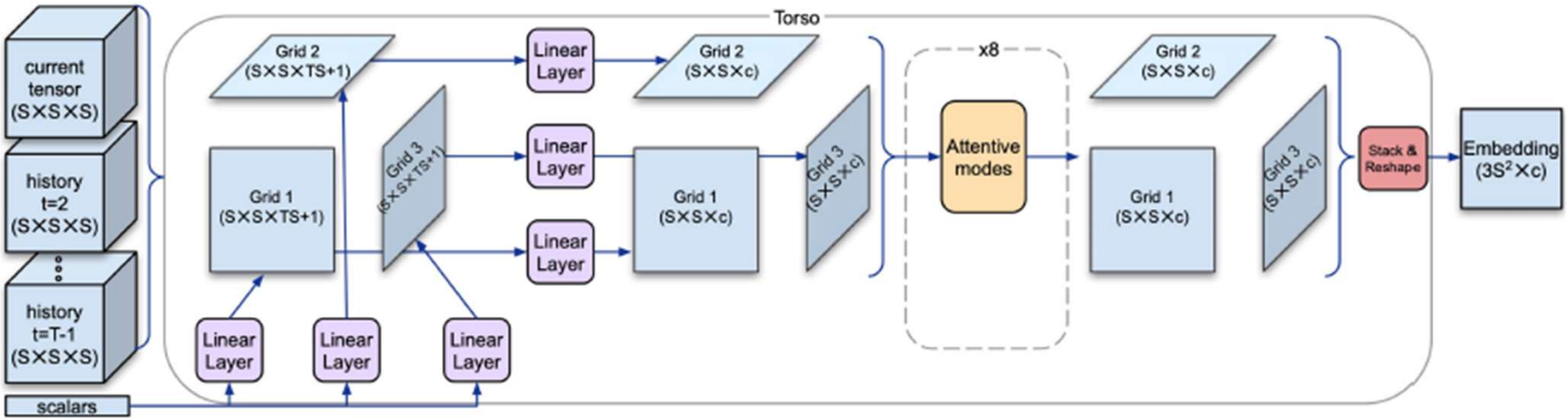
Training Time

- Learning:
  - TPU v3
  - 64 TPU cores
  - Train for 600,000 iterations
- Acting:
  - TPU v4
  - 1600 actors
- Training converges in 1 week

Network Architecture

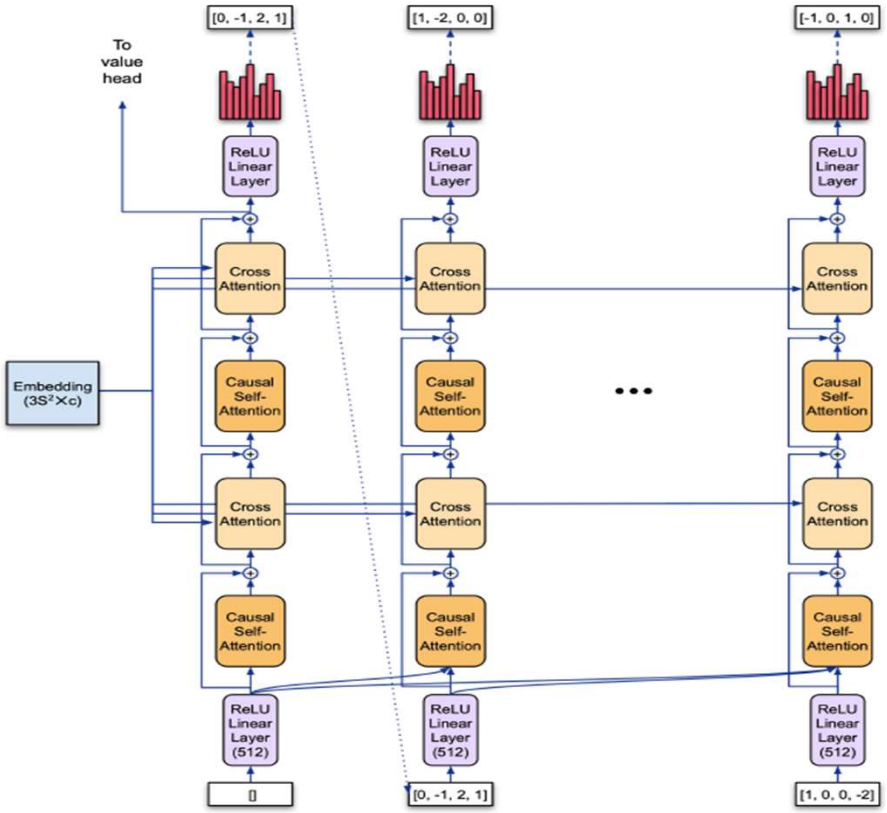


Torso

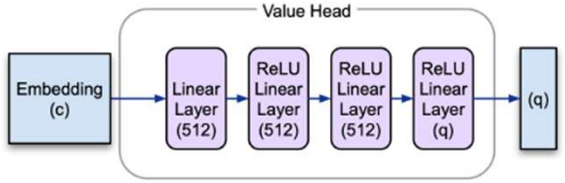

current tensor (S×S×S)
history t=2 (S×S×S)
history t=T-1 (S×S×S)
scalars
Torso
Grid 2 (S×S×TS+1)
Linear Layer
Grid 2 (S×S×c)
Grid 1 (S×S×TS+1)
Grid 3 (S×S×TS+1)
Linear Layer
Linear Layer
Grid 1 (S×S×c)
Grid 3 (S×S×c)
Linear Layer
Linear Layer
Linear Layer
x8
Attentive modes
Grid 2 (S×S×c)
Grid 1 (S×S×c)
Grid 3 (S×S×c)
Stack & Reshape
Embedding (3S²×c)

Policy and Value Heads

- Extensive use of Cross Attention



(b) Policy head diagram. Blocks in the same row share the learning weights.



(c) Value head diagram.

Thoughts

- A lot of inductive biases for matrix multiplication built into the system
  - Could help speed up learning
- Reward not just only win/loss/draw like AlphaZero, but more informative which is based on closeness to the goal state (and also compute time)
  - Similar to KataGo incorporating more rewards such as ownership loss and score belief loss to help the network focus on erroneous predictions
  - Easier to credit assign than just the final game outcome
- Attention mechanisms can help with credit assignment

Questions to Ponder

- Can we add more reward (e.g. intrinsic reward) to make the learning process more efficient?
- Can we sample the stored buffer better (e.g. hippocampal replay) to make the learning process more efficient?
- Could there be some form of algorithmic discovery whereby a sequence of actions lead to the elimination of one multiplication step, and we can apply this recursively?
- Is there a way to simplify TensorGame even more, by perhaps just considering fewer possibilities?
- How do we apply this to arbitrary one-player games?

FBHHRBNRSSSHK-Algorithm

Manuel Kauers and Jakob Moosbauer

Overview

- Human-based multiplication of matrices
- Achieves 95 multiplications (instead of 96 found by AlphaTensor) for multiplication of 5x5 matrices
- Used the scheme of AlphaTensor
- Apply recursively:
  - Have a schema of multiplications
  - Applying a sequence of transformations to eliminate one multiplication

Problem Set-up

3. Multiplication of 5×5 Matrices

Let

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} \end{pmatrix} \text{ and } B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix},$$

and

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \\ c_{5,1} & c_{5,2} & c_{5,3} & c_{5,4} & c_{5,5} \end{pmatrix} := AB$$

be matrices with coefficients in a ring R of characteristic 2.

Inclusion-Exclusion Matrices

We can then compute the entries of C from the entries of A and B by first setting

$$m_k = \left(\sum_{i=1}^{5}\sum_{j=1}^{5} \alpha_{i,j}^{(k)} a_{i,j}\right)\left(\sum_{i=1}^{5}\sum_{j=1}^{5} \beta_{i,j}^{(k)} b_{i,j}\right)$$

for $k = 1, \ldots, 95$ and then

$$c_{i,j} = \sum_{k=1}^{95} \gamma_{i,j}^{(k)} m_k$$

for $i, j = 1, \ldots, 5$, where the $\alpha_{i,j}^{(k)}, \beta_{i,j}^{(k)}, \gamma_{i,j}^{(k)}$ are as follows:

k	$((\alpha_{i,j}^{(k)}))_{i,j=1}^{5}$	$((\beta_{i,j}^{(k)}))_{i,j=1}^{5}$	$((\gamma_{i,j}^{(k)}))_{i,j=1}^{5}$
1	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$