[illegible]

matrix multiplication <1>

[illegible] matrix multiplication

= [illegible]

$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix}_{2\times 2}$

$n \times n$ matrix, we will need n^3 muls & $n^2(n-1)$ adds.

ex) $\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$ 에서 $n=2$

'$a_1 \cdot b_1 + a_2 \cdot b_3$' 에서 '+'는 1번 수행 $(n-1)$

만일 $n=3$ 이면,

$a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 \Rightarrow (n-1) = 2$번 덧셈 수행

$n^2 = \begin{bmatrix} (a_1) & (a_2) \\ (a_3) & (a_4) \end{bmatrix} = A$ A의 각 element인

'a_1, a_2, a_3, a_4' 각각 덧셈 하니까 n^2번

∴ 덧셈 총 $n^2 \cdot (n-1)$ adds

① $a_1 \cdot b_1 \oplus a_2 \cdot b_3$ → $(n-1)$번

② $a_1 \cdot b_2 \oplus a_2 \cdot b_4$

③ $a_3 \cdot b_1 \oplus a_4 \cdot b_3$

④ $a_3 \cdot b_2 \oplus a_4 \cdot b_4$

n^2번

$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times B = \begin{bmatrix} e & f \\ g & h \end{bmatrix} = C$

$a, b, c, d = \frac{N}{2} \times \frac{N}{2}$ sub matrix.

muls $= m^{3 \cdot} \cdot r^k$ ↑ side lengths $n = m \cdot 2^k$

two square matrices with

recursions의 수 = 곱셈의 수 (multiplications)

50년동안 = people 초점: exponent carving.

($w < 2.3754770$ 「JSC 1990」
$w < 2.3728596$ 「SODA, 2021」) 큰 개선 X.

「making this an 'algorithm'」

↓

발전이 크게 없어 관심을 받지 못했다 X.

problem ①

Focused on asymptotics — what is the best for a finite 'n'?

②

Focused on square matrices

↳ goal: $m \times n$, $n \times p$ matrix에 적용.

↓

강화 학습으로 해보자. (RL research).

why? used when a brute-force (사용이유) search is too expensive.

* Finding an optimal solution from a large search space by learning to search.

$a = f_\theta(s)$

↳ action ↳ 어떻게 바둑돌을 놓지.

↳ s=state ex) 바둑판 상황.

↳ How train a 'policy' network (that outputs next action given the state.)

* challenge = 각 상태가 좋은지 평가하는 방법이 부족. (lack a nice way)

→ 보상(reward is sparse).

「Monte carlo tree search = 알파고」

→ selection → 확장 → 시뮬레이션 → back propagation

「Tensor representations of MatMuls」

↳ one-player game = things are a bit easier, because we need to consider the opponent's action no longer.

「learning matrix multiplication algorithms」 (특징)

$$T_n = \sum_{r=1}^{R} u^{(r)} \otimes v^{(r)} \otimes w^{(r)}$$

<텐서 : tensor>.

$m_1 = (a_1 + a_4)\cdot(b_1 + b_4)$

$m_2 = (a_3 + a_4)\cdot b_1$

$m_3 = a_1(b_2 - b_4)$

$m_4 = a_4(b_3 - b_1)$

$m_5 = (a_1 + a_2)\cdot b_4$

$m_6 = (a_3 - a_1)(b_1 + b_2)$

$m_7 = (a_2 - a_4)(b_3 + b_4)$

$c_1 = m_1 + m_4 - m_5 + m_7$

$c_2 = m_3 + m_5$, $c_3 = m_2 + m_4$

$c_4 = m_1 - m_2 + m_3 + m_6$.

$$c_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

← b_1 → $a_1 \cdot b_1$

← b_2

← b_3 → $a_2 \cdot b_3$

↑ a_1 ↑ a_3

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

↑ ⓐ

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

↑ b

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

↑ c (m)

텐서 (tensor)

↳ 벡터의 개념을 확장한 기하학적인 양. 물리현상을 기술하기 위해 도입한 좌표계에는 무관계한 공간 또는 도형의 성질을 끌어져 추구해야 하기 때문에 이를 위해 만들어진 일반화된 좌표계이다.

tensor: 3차원 이상의 배열

① rank=0 「type= scalar」
ex) [1]

② vector = ex) [1,1] , rank=1.

③ matrix: rank=2, [[1,1], [1,1]]
= [[1,1], [1,1]]
→ 1행1열
→ 2행2열.

※ discussion

↳ ① numerical stability
② energy consideration
③ kernel-tizability

「Open Question」

① These 'better matmuls' are often less numerically stable

② write a separate code for each size?

※ 「Manuel Kauers, Jakob Moosbauer 논문」

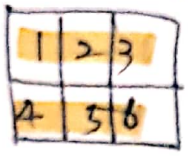
Tensor = 배열의 집합.

차원의 수 = rank

배열의 차원에 따라 불리는 이름이 달라진다.

↳ matrix = vector의 집합.

ex) [[1,2,3], [4,5,6]]



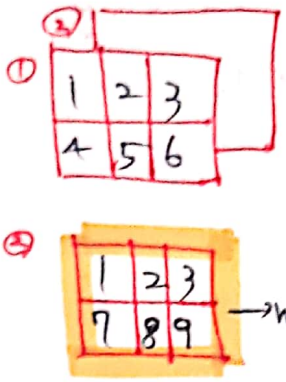

```
import numpy as np
x = np.array([ [1,2,3],
               [4,5,6],
               [7,8,9]])
```

「x.ndim = 2」

↳ vector 여러개가 모이면 2차원 데이터가 됩니다.

ex) tensor

```
import numpy as np
x = np.array([[[1,2,3],[4,5,6]],[[1,2,3],[7,8,9]]])
```




```
print(x)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [7 8 9]]]
```

→ matrix

↳ matrix의 집합인 tensor는 항연히 3차원 부터 시작하니, 최소 3이상의 수가 나올것입니다.

x.ndim ⇒ 「3」