Matricization of the Rubik's Cube

Stephen Huan

July 6, 2019

1 Rational

Apparently the Rubik's cube is an "algebraic group" (which I will not pretend to understand), and has certain mathematical properties. The most important in my opinion is *non-commutativity*, or when $a \times b \neq b \times a$. For a simple example, note that applying the sequence "R U" does not give the same result as applying "U R". Such "non-commutative algebra" is closely associated with things like Heisenberg's Matrix mechanics, a formulation of quantum mechanics mathematically identical to Schrödinger's wave mechanics or Hamilton's quaternions.

In my previous lecture I argued that a cubie-wise approach was simplier and more abstract than a sticker-wise approach. However, my cubie approach is 3x9x3, making it a tensor. Contrary to what Google wants you to believe, tensors are difficult to work with and do not "flow". A sticker-wise approach could be thought of as a 6x9 matrix, and moves as other transformation matrices such that if S is a cube state and R a possible turn of the cube $S' = SR$.

The advantages are numerous. Solving the cube becomes a matrix factorization problem - trying to decompose a single matrix which represents a a complex transition from scrambled to solved into the product of many move matrices. Matrix multiplication is a well-optimized operation in many different linear algebra libraries and is trivially parallelized.

2 Example

Suppose $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ is a cube state and $B = \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}$ is the cube state after a R move. We are looking for a matrix X such that $AX = B$. Multipying both sides on the left by A' yields $A'AX = A'B$ which means $X = A'B$. After computing the inverse $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. X can be thought of as a 2-swap which swaps the first row as well as the second row. Computing $AX^2 = A$, $AX^3 = AX$, etc.

However, if $B = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$ solving for X yields $X = \begin{bmatrix} -1 & 2 \\ 1.5 & -0.5 \end{bmatrix}$. By definition $AX = B$, however $AX^2 \neq A$ which would be expected of a swap. Further exponentiation

of X results in gibberish. This can be interepreted as the limitation of "learning" a transformation matrix: it cannot learn a single swap, only a two-swap.

The analogy to cubing is clear. Suppose we had a transformation matrix R which applies a R-move on a given state of the cube S. R2 would literally be R^2. R' would be R^3 and R4 would be R^4 as well as the identity matrix.

3 Experimentation

As stated previously, a stickermap is 6x9 or a non-square matrix. Therefore there does not exist a traditional inverse, and more general methods must be used. In particular I tried the Moore-Penrose inverse as well as the two one-sided inverses.

3.1 Solved Cube

Starting with the solved cube given by:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$

and a R-turn on the solved cube given by:

$$\begin{bmatrix} 0 & 0 & 5 & 0 & 0 & 5 & 0 & 0 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 2 & 2 & 4 & 2 & 2 & 4 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 0 & 4 & 4 & 0 & 4 & 4 & 0 \\ 5 & 5 & 2 & 5 & 5 & 2 & 5 & 5 & 2 \end{bmatrix}$$

using Srikar Gouru's code (ありがとう !).

I first computed the Moore-Penrose inverse but had no idea what to do with it. I then attempted to compute both one-sided inverses, but for an one-sided inverse to exist the rank of the matrix has to be the maximum possible for that size. A property of rank is that $\text{rank}(A) = \text{rank}(A^T)$, or that the columns rank is equal to the row rank. In this case, the maximum rank is 6. Since each row of the matrix is linearly dependent on each other row and each column is linearly dependent on each other column the rank of the solved cube is 1.

3.2 Random Permutation

To rectify the rank problem I permuted each $(i, j) \to (i', j')$ randomly.

```
def transform(mapping, m):
    new = [[0]*9 for i in range(6)]
    for i in range(6):
        for j in range(9):
            x, y = mapping[(i, j)]
            new[i][j] = m[x][y]
    return np.array(new)


def gen_random(m=solved):
    indexes = [(i, j) for i in range(6) for j in range(9)]
    trans = {}
    for i in range(6):
        for j in range(9):
            trans[(i, j)] = random.choice(indexes)
            indexes.remove(trans[(i, j)])
    return transform(trans, m), trans
```

To verify the inversability of the new matrices, I wrote a function.

```
def inversable(m, tol=2):
    m = np.array(m)
    if np.linalg.matrix_rank(m) == 6:
        return (round(np.linalg.det(m.T @ m), tol) != 0,
                round(np.linalg.det(m @ m.T), tol) != 0)
    return (False,)*2
```

It happened that the left inverse did not exist. The right inverse is defined by $A_r = A^T(AA^T)^{-1}$ with the result that $AA_r = AA^T(AA^T)^{-1} = I_6$. Given A as the solved cube state and B the cube after a R-turn, $AX = B$, so $X = A_rB$ is the transformation matrix.

However, repeated application of X was meaningless. Also, the permutation was completely ad-hoc and mathematically meaningless.

3.3 Distinct Swaps

Recalling the simple 2x2 example, certain swaps are possible and certain other swaps are impossible. I realized I could not possibly learn from the solved state $\rightarrow$ R-move off because only 12 stickers move (the entire red face is unchanged). I would therefore need a 20 sticker change, the maximum possible. To find such a state I defined the function rdiff which returns the number of stickers which change before and after doing a R move and applied my BFS defined in my earlier lecture.

```
c = cube.Cube()
states, alg = cube.solve(c, (None, lambda c: rdiff(c) >= 20), cube.HTM)
```

```
c.turn(alg)

print(rdiff(c))
print(inversable(c.to_face()))

a = np.array(c.to_face())
c.turn("R")
ap = np.array(c.to_face())

ar = a.T @ np.linalg.inv(a @ a.T)
R = ar @ ap
print(a @ R)
```

As a result of finding a 20 sticker difference I no longer needed to permute the matrix. However, the transition matrix remains meaningless.

4 Future Work

Perhaps transitions cannot be represented by matrices and have to be represented by tensors or quaternions. Whatever the case, the mathematical representation must be noncommutative. This also may be a fairly useless avenue of research.

Various vectorization approaches and neural networks to be done soon!