MatriQ project

Valter Uotila

QHack, February 2023

Introduction

In October 2022, Google's parent company Alphabet's DeepMind research team published an article *Discovering faster matrix multiplication algorithms with reinforcement learning* [2] which implemented a reinforcement learning -based approach called AlphaTensor to discover faster matrix multiplication algorithms. So the paper tackled the classical and important problem in computer science and machine learning: how to multiply matrices more efficiently? In my personal computer science research-oriented social media bubble, this paper started trending heavily because AlphaTensor actually found faster matrix multiplication algorithms.

After reading about the AlphaTensor's work and the problem, I noticed that it seems to have such a format that it could be turned into an interesting quantum computational optimization task. This QHack 2023 project, titled MatriQ, is a story of that work.

More precisely, this MatriQ project description document introduces the problem formulation at informal and formal levels. The informal level should be accessible to anyone familiar with basic arithmetic. It should motivate readers to understand that the matrix multiplication problem is strongly connected to the performance of any machine learning model. The formal problem definition utilizes tensors and divides the problem into different classes depending on the underlying scalar field. To keep this document self-contained, we shortly review the Quadratic Unconstrained Binary Optimization (QUBO) model and then formulate a matrix multiplication search algorithm as a QUBO minimization problem. After QUBO formulation, we discuss the implementation and review different technical methods to solve the problem. This part is not important regarding the solution to the problem but more about understanding the relationship between QUBO, QAOA, D-wave's Ocean package, Braket, and Pennylane. Finally, we discuss the results and outline future work. At the end of this document, I have included a short biography that tells about me and my research on quantum computing.

Problem definition

Problem informally

The standard 2×2 matrix multiplication is a relatively simple operation that we already learned in school:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \cdot e + b \cdot g & a \cdot f + b \cdot h \\ c \cdot e + d \cdot g & c \cdot f + d \cdot h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}. \tag{1}$$

From a computational perspective, $+$ operations are cheap, and $\cdot$ operations are more expensive. That's why we want to minimize the number of scalar multiplications $\cdot$ in the matrix multiplication algorithms. In other words, the number of scalar multiplications $\cdot$ gives us a concrete metric to determine when one matrix multiplication algorithm is better than another.

The fact that we might have not learned from school is that this method to multiply matrices is not the only one! Actually, the number of possible ways to multiply matrices is huge. We will discuss this matrix multiplication algorithm search space later.

The most optimal way to multiply 2×2 matrices is Strassen's algorithm [6]. As calculation 1 shows, the naive algorithm uses eight scalar multiplications $\cdot$ to multiply two 2×2 matrices. Strassen's algorithm performs the same multiplication using only seven scalar multiplications!

Before going into details, let's get our hands dirty and actually pay attention to Strassen's algorithm. The algorithm will play an important role in this quantum computing solution. Let the two matrices be as previously

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} e & f \\ g & h \end{bmatrix}.$$

We calculate the following intermediate values based on these matrices

$$\begin{aligned} m_1 &= (a + d) \cdot (e + h) \\ m_2 &= (c + d) \cdot e \\ m_3 &= a \cdot (f - h) \\ m_4 &= d \cdot (g - e) \\ m_5 &= (a + b) \cdot h \\ m_6 &= (c - a) \cdot (e + f) \\ m_7 &= (b - d) \cdot (g + h). \end{aligned}$$

Calculating the number of scalar multiplications $\cdot$ reveals that we have used only seven of them. Now Strassen's result [6] indicates that the result of the matrix multiplication is

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 - m_2 + m_3 + m_6 \end{bmatrix}.$$

All in all, we used only seven scalar multiplications $\cdot$. If you are developing a machine learning model and training requires matrix multiplications using 2×2 matrices, you definitely will save a lot of computational resources and time by using Strassen's algorithm. But we haven't really heard of ML models using as small matrices as 2×2, so the next question we want to ask is: does this algorithm scale easily? Unfortunately, the answer seems to be no. For example, we do not know the optimal matrix multiplication algorithm for 3×3 matrices. Then we curious people can continue asking questions:

1. How can we find faster multiplication algorithms for $n \times n$ cases than the naive algorithm?
2. How should we formalize the problem in a way that we would be able to use computers and optimization algorithms (or reinforcement learning as AlphaTensor) to discover faster matrix multiplication algorithms?
3. How could we prove that the algorithm is optimal?

The second question has a good answer which we will discuss next. The third question is out of the scope of this project, but it is interesting to keep it in mind, especially because Strassen's algorithm is proved to be the optimal [6]. Finally, the first question is related to the quantum computational part of this project.

Problem formally

Matrix multiplication algorithms as tensor decompositions

This part follows and explains the problem formulation in [2]. More detailed descriptions can be found in [1, 3, 4].

If we focus on multiplying $n \times n$ matrices, we obtain that matrix multiplication is a bilinear function

$$M_n \colon \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}, \quad M_n(A,B) = AB.$$

This bilinear operator is already defined by a tensor (cf. linear maps are defined by matrices). Now the core question is to find a tensor decomposition

$$M_n(A,B) = \sum_{i=1}^{r} f_i(A) g_i(B) W_i,$$

with linear mappings $f_i, g_i \colon \mathbb{R}^{n\times n} \to \mathbb{R}$ and $W_i \in \mathbb{R}^{n\times n}$. The index r in the sum gives the number of required scalar multiplications (for example, seven in the case of Strassen's algorithm) to perform the matrix multiplication. Any decomposition provides one method to multiply matrices. The difficulty of the problem from this angle is to find a decomposition whose length is as short as possible, i.e., the index value r is smaller than the previously best-known value.

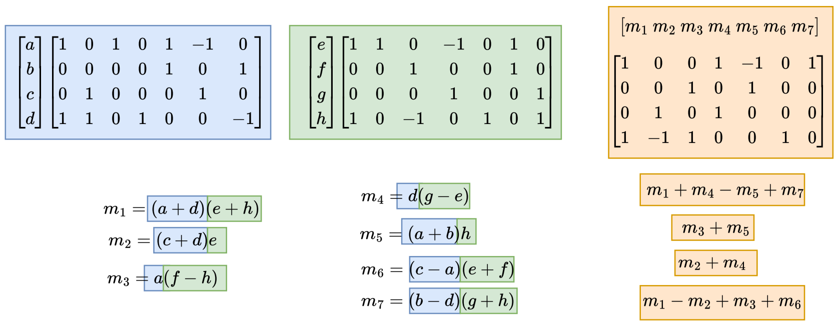
$\begin{bmatrix}a\\b\\c\\d\end{bmatrix}\begin{bmatrix}1&0&1&0&1&-1&0\\0&0&0&0&1&0&1\\0&1&0&0&0&1&0\\1&1&0&1&0&0&-1\end{bmatrix}$
$\begin{bmatrix}e\\f\\g\\h\end{bmatrix}\begin{bmatrix}1&1&0&-1&0&1&0\\0&0&1&0&0&1&0\\0&0&0&1&0&0&1\\1&0&-1&0&1&0&1\end{bmatrix}$
$[m_1\ m_2\ m_3\ m_4\ m_5\ m_6\ m_7]$
$\begin{bmatrix}1&0&0&1&-1&0&1\\0&0&1&0&1&0&0\\0&1&0&1&0&0&0\\1&-1&1&0&0&1&0\end{bmatrix}$
$m_1 = (a+d)(e+h)$
$m_2 = (c+d)e$
$m_3 = a(f-h)$
$m_4 = d(g-e)$
$m_5 = (a+b)h$
$m_6 = (c-a)(e+f)$
$m_7 = (b-d)(g+h)$
$m_1 + m_4 - m_5 + m_7$
$m_3 + m_5$
$m_2 + m_4$
$m_1 - m_2 + m_3 + m_6$

Figure 1: Optimal tensor decomposition for Strassen's algorithm [2]

The paper [2] provides very illustrative visualization of the mappings $f_i, g_i\colon \mathbb{R}^{n\times n} \to \mathbb{R}$ and $W_i \in \mathbb{R}^{n\times n}$ when $n = 2$ (i.e. for Strassen's algorithm). The modified visualization is in Figure 1.

We proceed in a way that we search for vectors $u_i, v_i, w_i \in \mathbb{R}^n$ so that

$$M_n = \sum_{i=1}^{r} u_i \otimes v_i \otimes w_i.$$

For example, Figure 1 gives us the vectors $u_1 = [1,0,0,1]^\top$, $v_1 = [1,0,0,1]^\top$, $w_1 = [1,0,0,1]^\top$ which corresponds to the first columns of the matrices in Figure 1. The second column provides the vectors u_2, v_2, w_2, etc.

In this work, we assume that $\mathbb{R} = \mathbb{F}_2 = \{0,1\}$ is the finite two-element field where addition is defined by addition modulo two and multiplication follows the multiplication of 0 and 1 in reals. This assumption was also made in [2] for part of the cases they studied but it simplifies the calculations at least in our quantum computational approach.

Reinforcement approach by AlphaTensor

AlphaTensor [2] approaches the faster matrix multiplication search as a reinforcement learning problem, modeling the environment as a single-player game.

In each step i of the game, the player chooses three vectors u_i, v_i, w_i. The tensor M_i is updated by subtracting the resulting tensor:

$$M_i := M_{i-1} - u_i \otimes v_i \otimes w_i.$$

The player aims to reach the zero tensor by applying the smallest number of moves. When the player reaches the zero tensor, the sequence of selected vectors automatically satisfies

$$M_n = \sum_{i=1}^{r} u_i \otimes v_i \otimes w_i.$$

This guarantees that the resulting matrix multiplication algorithm is correct. Approaching the problem from the reinforcement angle makes sense because reinforcement learning is designed to maximize long-term reward.

QUBO formulation

This section describes how we tackle the matrix multiplication algorithm discovery problem from the QUBO perspective (aiming to minimize objective function or ”energy”). Matrix discovery with quantum annealing is divided into multiple QUBO executions. I interpret that this method makes the algorithm hybrid because it consists of varying execution between quantum computer and classical problem formulation which depends on the previous solution obtained from a quantum computer.

Let us take the ”game” perspective from AlphaTensor’s work. The game consisted of steps where the player chooses vectors and creates the tensor products. Now the player is the quantum annealer. The quantum annealer selects the vectors so that they aim to minimize a suitable objective function. Recall that the reinforcement algorithm aimed to reach the zero tensor. This means that, in some sense, the vectors should be selected so that the number of ones in the tensor will decrease. Thus it makes sense that the energy or the penalty value where the objective function maps the binary variable configurations depends on the number of ones in the outputted tensor.

Let us fix the step i and study the case of 2×2 matrices. The approach generalizes easily to $n \times n$ matrices. We start by defining the binary variables for this fixed step i. The binary variables are simply the elements of the following vectors:

$$\begin{aligned} \mathbf{u}_i &= [u_1, u_2, u_3, u_4]^\top \\ \mathbf{v}_i &= [v_1, v_2, v_3, v_4]^\top \\ \mathbf{w}_i &= [w_1, w_2, w_3, w_4]^\top. \end{aligned}$$

So, after executing the algorithm, we obtain a configuration, for example, $w_1 = 0, w_2 = 1, w_3 = 0, w_4 = 0$. The (first) idea of constructing the objective function is very simple. We calculate the tensor product of the binary variable vectors

$$\mathbf{u}_i \otimes \mathbf{v}_i \otimes \mathbf{w}_i,$$

take the previously obtained tensor M_{i-1} and calculate the subtract

$$M_{i-1} - \mathbf{u}_i \otimes \mathbf{v}_i \otimes \mathbf{w}_i.$$

This follows the idea represented in AlphaTensor [2] paper.

If we calculate the tensor $\mathbf{u}_i \otimes \mathbf{v}_i \otimes \mathbf{w}_i$ explicitly, we find out that, for example, in the tensor’s index $(1, 1, 1)$, we have the element $u_1 v_1 w_1$. This type of triplet is not an allowed term by the definition of QUBO. There are at least two ways to fix this with auxiliary variables. I have implemented one version, but now I started thinking it is wrong and I should use this method[1] because it is more variable efficient. (Anyway, I will explain what I have done and fix it later.)

We need to turn all of these triples $u_1v_1w_1$ into the quadratic format by creating auxiliary variables $x_{(u_1v_1)}$, $x_{(u_1w_1)}$ and $x_{(v_1w_1)}$. After introducing the auxiliary variables, we rewrite the problematic part by changing $u_1v_1w_1$ into sum

$$x_{(u_1w_1)}v_1 + x_{(v_1w_1)}u_1 + x_{(u_1v_1)}w_1.$$

The idea is that the previous function and the triple $u_1v_1w_1$ have the same "energy landscape". Besides, we add constraints that have the following type

$$(2x_{(u_1w_1)} - u_1 - w_1)^2 = 4x_{(u_1w_1)} + u_1 + w_1 - 4x_{(u_1w_1)}u_1 - 4x_{(u_1w_1)})w_1 + 2u_1w_1.$$

This constraint encodes the fact that $x_{(u_1w_1)} = 1$ iff $u_1 = w_1 = 1$. We perform this rewriting operation for each element in the tensor $\mathbf{u}_i \otimes \mathbf{v}_i \otimes \mathbf{w}_i$.

The tensor we obtained at step $i-1$ is M_{i-1}. Because we operate in the field $\mathbb{F}_2$, it contains only 1s and 0s. Thus we obtain terms like $u_1v_1w_1$ (if there is zero in tensor M_{i-1}) or $1 - u_1v_1w_1$ (if there is one in tensor M_{i-1}). The outcome is a tensor that contains binary variables in its elements. Now we "flatten" the tensor, meaning we just collect all the terms from the tensor and sum them together to form the final objective function. This objective function is minimized on a quantum annealer. From the solution, we obtain the tensor M_i, which we will utilize on the next step $i+1$.

The algorithm relies on the fact that, for some magical reason, minimizing objective function at each step would minimize the objective function over the whole process. I do not see a reason why this should be true in general. But this magical property is satisfied in 2×2 case if we start the algorithm from a suitable place in the solution space. That is one of the most surprising results that I have seen in my personal research ever. I will discuss that in the results section.

Implementation

I have implemented the initial version of this algorithm in the Jupyter notebooks 2×2.ipynb, 3×3.ipynb and 4×4.ipynb. The code in 2×2.ipynb shows a full example and two other notebooks 3×3.ipynb and 4×4.ipynb rely on utils.py and qubos.py packages. The functions in the code perform the QUBO formulation as I described in the previous section.

During this hackathon, I plan to translate D-wave's Ocean QUBOs into Hamiltonians in Pennylane (if possible), which is a more technical coding problem but something I have wanted to study and do. I have done the same between Qiskit and Ocean.

I also wanted to use Nvidia (because of the prices, of course!) to see if there are some possibilities. I have experience with Cuda and used GPU-accelerated

[1] `https://docs.dwavesys.com/docs/latest/handbook_reformulating.html#non-quadratic-higher-degree-polynomials`

JAX/jit to simulate circuits in my work. Actually, I must have run that code on my GPU for hours. It is difficult to say if there was any benefit but interesting coding problem to make GPU frameworks to work with quantum circuits.

Outcomes and future work

Future work could include at least deeper studies on the following topics.

- Getting rid of steps. Minimization should be performed over all the vectors v_i, u_i, w_i and not just over a single step. Now, this algorithm does not focus on any type of long-term minimization. In other words, we could fix the number of vectors we want to choose. For example, one less than the previously best-known algorithm performs. Then we can create the objective function over all these vectors. I need to study this further with the ideas presented here[2].

Biography

I am Valter Uotila, a soon-second-year Ph.D. student researching quantum computing applications for databases. My last year's QHack project turned into a paper *Quantum Annealing Method for Dynamic Virtual Machine and Task Allocation in Cloud Infrastructures from Sustainability Perspective*, which is accepted to be published at a small workshop at ICDE 2023 conference in April. It seems that there might be a few technical problems with the workshop, but anyway, that project eventually led to very interesting work and results, which will definitely be properly developed further!

Besides, I am about to publish work on estimating metrics for SQL queries in relational databases using a quantum machine learning model from quantum natural language processing [5]. We are also organizing a quantum computing for databases workshop at VLDB 2023 conference, and our tutorial on quantum machine learning has been accepted to SIGMOD 2023. Both of the conferences are the leading database conferences. There is a lot of interesting quantum computing going on and excited to be part of it!

References

[1] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[2] 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

[2] `https://docs.dwavesys.com/docs/latest/handbook_reformulating.html#non-quadratic-higher-degree-polynomials`

Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, Oct 2022.

[3] J. M. Landsberg. *Geometry and Complexity Theory*. Cambridge University Press, 1 edition, Sep 2017.

[4] Lek-Heng Lim. Tensors in computations. *Acta Numerica*, 30:555–764, May 2021. arXiv:2106.08090 [cs, math].

[5] Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. Qnlp in practice: Running compositional models of meaning on a quantum computer. *arXiv:2102.12846 [quant-ph]*, Feb 2021. arXiv: 2102.12846.

[6] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug 1969.