41076: Methods in Quantum Computing

Maria Kieferova

July 2020

1 Hamiltonian simulation

Given an initial state of a system $|\psi(0)\rangle$ and a Hamiltonian H, our goal is to simulate the time evolution $|\psi(0)\rangle \to e^{-iHt}|\psi(0)\rangle$. The goal of Hamiltonian simulation is to design a circuit U consisting of gates and oracles that approximates the time evolution up to an error ϵ such that

$$\left\| U - e^{-iHt} \right\|_2 < \epsilon, \tag{1}$$

where $\|\cdot\|_2$ is the spectral norm.

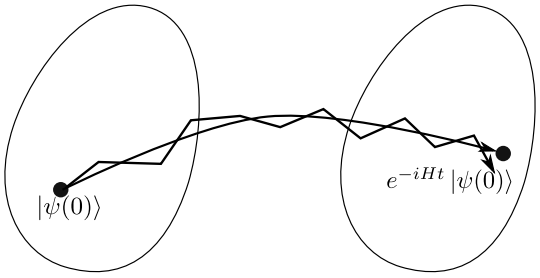

Figure 1: Hamiltonian simulation approximates the time evolution by a series of digital operations.

Quantum systems are fundamentally difficult to simulate; dynamics of quantum systems is a BQP-hard (or BQP complete for Hamiltonians with natural restrictions). Except for a few special cases, the complexity of the best known classical algorithms grows exponentially with the number of qubits. As such, simulation of quantum dynamics is a field where quantum computers can quickly outperform classical ones. In fact, the time evolution of quantum systems was the original application for quantum computers suggested by Feynman.

In the simplest scenario, we assume that the simulated Hamiltonians are given in the form $H = \sum_{j=1}^{m} H_j$, where each H_j is local and $e^{-iH_j t}$ can be

implemented directly for arbitrary t. A common case is when these H_j are Pauli matrices.

The simplest quantum simulation algorithms rely on the Lie-Trotter formula

$$\lim_{r\to\infty}\left(e^{A/r}e^{B/r}\right)^r = \lim_{r\to\infty}\left((1+\frac{A}{r})(1+\frac{B}{r})\right)^r \tag{2}$$

$$= \lim_{r\to\infty}\left(1+\frac{A+B}{r}+\frac{AB}{r^2}\right)^r \tag{3}$$

$$= \lim_{r\to\infty}\left(1+\frac{A+B}{r}\right)^r \tag{4}$$

$$= e^{A+B} \tag{5}$$

Since the individual terms in the Hamiltonian typically not commute, decomposing an exponential of a sum into a finite sum of exponentials will lead to errors.

Exercise: Prove that $\left\|e^{A+B}-\left(e^{A/r}e^{B/r}\right)^r\right\| \in \mathcal{O}\left(\frac{\|A+B\|t^2}{r}\right)$.

Next, we can recursively that for $H=\sum_{j=1}^m H_j$, one can decompose the evolution with respect to H into the evolution with respect to each H_j as

$$\widetilde{U} = \left(e^{-iH_1t/r}e^{-iH_2t/r}\dots e^{-iH_mt/r}\right)^r + \mathcal{O}(\|H\|\,t^2/r). \tag{6}$$

Thus, if we are willing to tolerate error at most ϵ, we need to perform $\mathcal{O}(\frac{\|H\|t^2}{\epsilon})$.

Up to now, we assumed that we know how to implement each e^{-iH_jt} directly. Let us now show how to implement them in some simple cases.

In the simplest case, H_j is a Pauli matrix Z acting on the jth qubit. The Hamiltonian evolution is then a Z-rotation on the jth qubit.

$$e^{-itZ} = e^{-it}\left|0\right\rangle\left\langle 0\right| + e^{it}\left|1\right\rangle\left\langle 1\right| \tag{7}$$

Next, we show how to simulate a tensor product of Zs

$$H = Z_1\otimes Z_2\otimes\cdots\otimes Z_n. \tag{8}$$

We first prove the following identity. For an arbitrary unitary U:

$$Ue^{-iHt}U^\dagger = U\sum_{k=0}^\infty\frac{(-iHt)^k}{k!}U^\dagger \tag{9}$$

$$= UU^\dagger - iUHU^\dagger t + i^2UH(UU^\dagger)Ht^2U^\dagger$$
$$- i^3UH(UU^\dagger)H(UU^\dagger)HU^\dagger t^2+\dots \tag{10}$$

$$= \sum_{k=0}^\infty\frac{(-iUHU^\dagger t)^k}{k!} \tag{11}$$

$$= e^{-iUHU^\dagger t}. \tag{12}$$

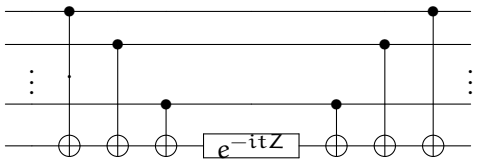

Figure 2: Simulating a tensor product of Pauli Zs.

We can implement $e^{-iZ_1\otimes\cdots\otimes Z_n t}$ using the circuit in Fig. 2

The correctness of the circuit can be showed through induction. We already proved the first step in (7). In the inductive step, we conjugate an existing circuit by CNOTs. We can rewrite $\mathrm{CNOT} e^{-itZ_1\otimes Z_2\otimes\cdots\otimes Z_{n-1}}\mathrm{CNOT}$ as:

$$(\mathbb{I}\otimes|0\rangle\langle 0| + X\otimes|1\rangle\langle 1|)\left(\mathbb{I}\otimes e^{-itZ_1\otimes\cdots\otimes Z_{n-1}}\right)(\mathbb{I}\otimes|0\rangle\langle 0| + X\otimes|1\rangle\langle 1|) \tag{13}$$

$$=\otimes\, e^{-itZ_1\otimes\cdots\otimes Z_{n-1}}\otimes|0\rangle\langle 0| + Xe^{-itZ_1\otimes\cdots\otimes Z_{n-1}}X\otimes|1\rangle\langle 1| \tag{14}$$

$$=e^{-itZ_1\otimes\cdots\otimes Z_{n-1}}\otimes|0\rangle\langle 0| + e^{itZ_1\otimes\cdots\otimes Z_{n-1}}\otimes|1\rangle\langle 1| \tag{15}$$

$$=e^{-itZ_1\otimes Z_2\otimes\cdots\otimes Z_{n-1}\otimes Z_n} \tag{16}$$

Simulating other Paulis is possible by changing the basis. Recall that $e^{-itUHU^\dagger} = Ue^{-itH}U^\dagger$. We can then use the identities

$$X = \mathrm{Had}\; Z\; \mathrm{Had} \tag{17}$$

$$Y = S^\dagger \mathrm{Had}\; Z\; \mathrm{Had}\; S^\dagger \tag{18}$$

We used Had instead of H to denote the Hadamard gate since we reserved the symbol H for Hamiltonians.

This allows us to simulate evolution according to any Pauli. For example, we can simulate the evolution according to the $H = X\otimes Y\otimes Z$ according to the circuit in Fig. 3.

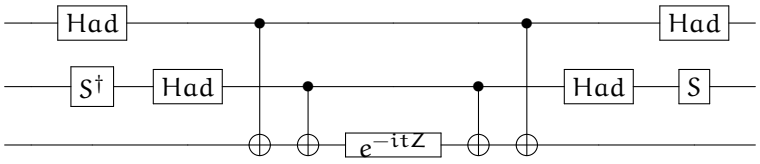

Figure 3: X and Y Paulis can be simulated by a change of basis. The circuit above depicts the simulation according to $e^{-itX\otimes Y\otimes Z}$.

Now we know how to simulate any Hamiltonian that is a sum of Paulis. The complexity of the algorithm for simulating a Hamiltonian $H = \sum_{l=0}^{L-1} P_n$ where

P is a Pauli on n qubits is

$$\mathcal{O}(\frac{Lt^2 n}{\epsilon}) \tag{19}$$

However, there is still a need for more efficient algorithms. First, the number of terms L needed to decompose a Hamiltonian into a sum of Paulis can be exponentially large. Second, the scaling in terms of t and ϵ is quite poor.

Other Hamiltonian simulation algorithms allow for different decomposition of Hamiltonians. A popular one is decomposing a sparse matrix into 1-sparse matrices which can be then simulated directly. Another one is decomposition into a linear combination of unitaries (LCU) which a generalization of the Pauli decompositions.

Exercise: Let ρ, σ be density matrices, S the SWAP operator and Tr_p partial trace over the first variable. Show that

$$\mathrm{Tr}_p[e^{-iS\Delta}\rho \otimes \sigma e^{iS\Delta}] = e^{-i\rho\Delta}\sigma e^{i\rho\Delta} + \mathcal{O}(\Delta^2)$$

In these more general cases the Hamiltonian is accessed through an oracle. One type of oracular access is particularly common when the Hamiltonian (in a computational basis) is given by a sparse matrix. We say a Hamiltonian is *row-d-sparse* if each row has at most d non-zero entries. If there is an efficient procedure to locate these entries we moreover say that the Hamiltonian is *row-computable*. In this case, one can efficiently construct oracles

$$O_{loc}\,|r, k\rangle = |r, k \oplus l\rangle \tag{20}$$

$$O_{val}\,|r, l, z\rangle = |r, l, z \oplus H_{r,l}\rangle\,. \tag{21}$$

Oracle O_{loc} locates the position l of the k-th non-zero element in row r. The oracle O_{val} then gives the value of the matrix element $H_{r,l}$. We compute the cost of algorithms in terms of the number of queries to these oracles.

It is possible to construct different oracles. Any Hermitian matrix can be decomposed into a sum of unitaries

$$H = \sum_{l=0}^{L-1} \alpha_l V_l, \tag{22}$$

where for each l, $\alpha_l \leqslant 0$ and H_l is a unitary matrix $\|H_l\| = 1$. This decomposition can be efficiently implemented for sparse Hamiltonians. The coefficients α_l and unitaries H_l can be accessed through oracles

$$O_\alpha\,|l, z\rangle = |l, z \oplus \alpha_l\rangle \tag{23}$$

$$O_{V_l}\,|l, \psi\rangle = V_l\,|l, \psi\rangle\,, \tag{24}$$

or, in some cases, described classically.

1.1 Further reading

Efficient quantum algorithms for simulating sparse Hamiltonians
G. Ahokas, R. Cleve, B.C. Sanders, D.W. Berry
arXiv:quant-ph/0508139

Simulating Hamiltonian dynamics with a truncated Taylor series
D. W. Berry, A. M. Childs, Richard Cleve, R. Kothari, R. D. Somma
arXiv:1412.4687

Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics
A Gilyén, Y Su, GH Low, N Wiebe
arXiv:1806.01838

2 Linear Systems

An important application of Hamiltonian simulation is "solving" systems of linear equations. The definition of solving is different for quantum and classical algorithms. While the classical algorithm outputs the solution of the linear system, quantum algorithms encode the solution into a quantum state. We will first review the classical approaches to solving systems of equations and then explain the corresponding quantum algorithms.

A system of equations can be written in a standardized form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_N &= b_1 \\ a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_N &= b_1 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_N &= b_n, \end{aligned} \tag{25}$$

or a matrix equation

$$\mathbf{A}\boldsymbol{x} = \mathbf{b} \tag{26}$$

where A encode the coefficients in the equations, the vector $\mathbf{x}$ are the unknowns and $\mathbf{b}$ stores the right hand side of equations. The solution then can be written as

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{27}$$

if the matrix A is indeed invertable. We will for now assume that this is the case and consider the linear system problem to be equivalent to matrix inversion.

One of the most well-known algorithms is Gaussian elimination with complexity $\mathcal{O}\left(N^3\right)$. While the complexity is polynomial in the number of equations and therefore "efficient", cubic runtime is impractical for large systems. Therefore, there is a lot of interest in finding more efficient exact and approximation algorithms.

The first quantum linear systems algorithm is referred to as HHL after the initials of its creators - Harrow, Hassidim and Lloyd. The linear system

algorithm makes efficient use of phase estimation. It is sufficient to consider Hermitain matrix; if the matrix A is not hermitian, a Hermitian matrix can be created as $H = \begin{pmatrix} \cdot & A \\ A^\dagger & \cdot \end{pmatrix}$ and take the right hand side into $\begin{pmatrix} b \\ 0 \end{pmatrix}$. The solution will this new system will then be $\begin{pmatrix} 0 \\ x \end{pmatrix}$. Without the lost of generality, we will from now on assume that A is hermitian.

Let $|\alpha_i\rangle$ and λ_i be the eigenvectors and eigenvalues of A and $|b\rangle = \sum_i \beta_i |\alpha_i\rangle$ be a decomposition of the right-hand side into this basis.

Using Hamiltonian simulation (with t=1) we and phase estimation we can prepare

$$|b\rangle |0\dots0\rangle |0\rangle \to \sum_i \beta_i |\alpha_i\rangle \left|\widetilde{\lambda}_i\right\rangle |0\rangle . \tag{28}$$

Next, we apply a rotation conditioned on $\left|\widetilde{\lambda}_i\right\rangle$ to prepare

$$\sum_i \beta_i |\alpha_i\rangle \left|\widetilde{\lambda}_i\right\rangle \left(\sqrt{1 - \frac{C^2}{\widetilde{\lambda}_i^2}} |0\rangle + \frac{C}{\widetilde{\lambda}_i} |1\rangle\right), \tag{29}$$

where $C = \mathcal{O}(1/\kappa)$. The phase estimation is the uncomputed, erasing the second register, and we are left with

$$\left(\sqrt{1 - \frac{C^2}{\widetilde{\lambda}_i^2}} \sum_i \beta_i |\alpha_i\rangle |0\dots0\rangle |0\rangle + \frac{C}{\widetilde{\lambda}_i} \sum_i \beta_i |\alpha_i\rangle |0\dots0\rangle |1\rangle\right). \tag{30}$$

In case when the last qubit is in state 1, the resulting state on the first register is proportional to $\sum_i \lambda_i^{-1} \beta_i |\lambda\rangle_i$

Overall, the complexity of the algorithm is $\widetilde{\mathcal{O}}\left(\log n \frac{s^2\kappa^2}{\epsilon}\right)$.

While HHL achieves exponential improvement in n when naively compared to classical algorithms, its scaling in the error ϵ and the conditioning number κ is not optimal.

However, this comparison is not completely fair. First, the algorithm is limited to matrices that can be effectively encoded in a quantum computer which are usually either sparse or low-rank matrices. Second, the state b also needs to be prepared effectively on a quantum computer. Lastly, the output of HHL is a quantum state and outputting it (i.e., writing it on a paper) would result in an exponential overhead and worse performance than most classical algorithms. Instead, the task of the quantum computer is often only to sample from the solution. A recent result managed to "de-quantize" many applications of HLL and showed that an algorithm for "solving" linear systems with a low-rank matrix and sampling from the solution can be simulated classically.

2.1 Further reading

Quantum algorithm for solving linear systems of equations

AW Harrow, A Hassidim, S Lloyd
arXiv:0811.3171

Quantum Machine Learning Algorithms: Read the Fine Print S Aaronson
https://www.scottaaronson.com/papers/qml.pdf

Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning
N Chia, A Gilyén, T Li, H Lin, E Tang, C Wang
arXiv:1910.06151

3 Quantum Machine Learning (QML)

HLL can be applied to various quantum machine learning problems. QML build on the intuition that most of machine learning is some form of linear algebra in high-dimensional spaces and quantum computing is good at performing algebra at high-dimensional spaces[1]. However, we will see that that obtaining quantum speedup for machine learning algorithms is not so straightforward.

3.1 Least-squares fitting

The first application is to apply HLL algorithm for data fitting. Our goal is to to find a parameters $\boldsymbol{\lambda}$ such that a parametric function $f(\boldsymbol{x}, \boldsymbol{\lambda})$ will provide an optimal fit for N datapoints $\{x_i, y_i\}$.

The function is constrained to be linear in the fit parameters $\boldsymbol{\lambda} \in \mathbb{C}^M$, but it can be non-linear in $\boldsymbol{x}$. For example, we can encode $f(\boldsymbol{x}, \boldsymbol{\lambda}) = a + bx + cx^2$ as $\boldsymbol{\lambda} = (a, b, c)$ and $\boldsymbol{x} = (1, x, x^2)$.

To goal of the algorithm is then find the parameters $\boldsymbol{\lambda}$ to minimize

$$E = \sum_{i=1}^{N} |f(x_i, \boldsymbol{\lambda}) - y_i|^2 = |\boldsymbol{F}\boldsymbol{\lambda} - \boldsymbol{y}|^2. \tag{31}$$

Here we defined $\boldsymbol{F}_{ij} = f_j(x_i)$ to be an $N \times M$ matrix. Note that $\boldsymbol{F}$ is generally not invertable, however, we can assume that $\frac{1}{\kappa} \leqslant \left\|\boldsymbol{F}^\dagger \boldsymbol{F}\right\| < 1$. One can then show that the minimum of (31) is achieved by

$$\boldsymbol{\lambda} = \boldsymbol{F}^+ \boldsymbol{y} = \left(\boldsymbol{F}^\dagger \boldsymbol{F}\right)^{-1} \boldsymbol{F}\boldsymbol{y}, \tag{32}$$

where F^+ is Moore–Penrose pseudoinverse.

The quantum version of least squares fitting assumes that y is encoded into a quantum states $|y\rangle = \sum_{j=M+1}^{M+N} y_j |j\rangle / \mathrm{norm}\boldsymbol{y}$. Similarly, the output $|\lambda\rangle$ will be also given only as a quantum state. Additionally, we have an oracular access to

[1] From Seth Lloyd, paraphrased

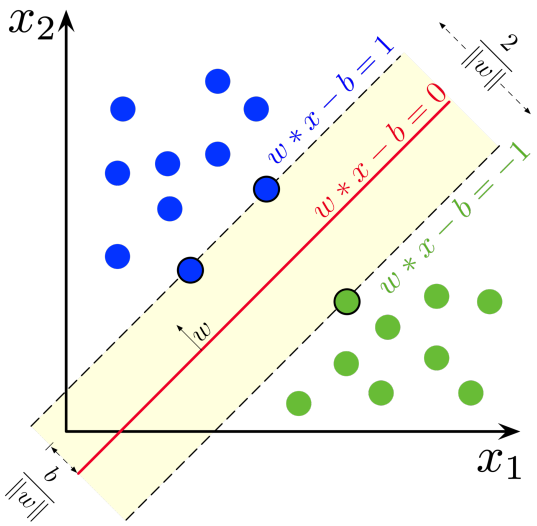

Figure 4: SVM for 2-dimensional data. The red line maximizes the separation between the classes. Source: Wikipedia

$\tilde{F} = \begin{pmatrix} \cdot & F \\ F^\dagger & \cdot \end{pmatrix}$. For an efficient algorithm, $\tilde{F}$ must belong into a class of simulable Hamiltonians such as sparse or low-rank ones.

The algorithm for least square fitting is then a technical modification of the HLL algorithm. First, we need to prepare $F|y\rangle$. This can be accomplished using HHL but though Hamiltonian evolution and phase estimation just as in HLL but instead of diving by the eigenvalues we will multiply by them. The inverse of $F^\dagger F$ is accomplished though HHL as well. The resulting query complexity of the algorithm is $\tilde{O}\left(\log(N)s^4\kappa^6/\epsilon\right)$ where s is the sparsity and the notation $\tilde{O}$ suppresses log-factors.

3.2 Quantum Support Vector Machine

Support vector machine (SVM) is a supervised machine learning algorithm for data classification. The algorithm is given a set of data where each data point belongs to one of two categories (green or blue). The goal of the algorithm is to find a hyperplane that would separate the green points from the blue ones. Since there are many such hyperplanes, we aim to find one that separates them with the largest margin, see Fig. 4. This is an example of a non-probabilistic binary linear classifier and although SVMs can be generalized to find non-linear classifiers, we will focus on this simple example.

We know from analytic geometry that a hyperplane is a set of points $\mathbf{x}$ that satisfy

$$\boldsymbol{w} \cdot \mathbf{x} - b = 0, \tag{33}$$

where $\boldsymbol{w}$ is the normal vector. Note that we do not require w to be normalized –

it's norm will determine the side of the boundary. We will require that all the blue points $\mathbf{x}$ satisfy

$$\boldsymbol{w} \cdot \mathbf{x} - b > 1, \tag{34}$$

and for each the green point

$$\boldsymbol{w} \cdot \mathbf{x} - b < -1, \tag{35}$$

as in Fig. **??**. If we use $y = +1$ to label all the blue data and $y = -1$, we can consolidate (34) and (35) as

$$y_i(\boldsymbol{w} \cdot \mathbf{x_i} - b) > 1, \ \ \forall(x_i, y_i) \in \text{training set}, . \tag{36}$$

The goal of the training is then to minimize $\|w\|$ subject to the constrain (36).

To turn SVM into a quantum algorithm, we use the usual tricks, turning vectors into quantum states, and reformulate (36) and reformulating SVM as a linear system that can be solved using HLL.

3.3 The weaknesses of quantum machine learning

We saw two algorithms that appear to give exponential speedup over their classical counterparts. Indeed, in our setting, the runtimes of the algorithms are faster than even writing the complete input. This is because the assumptions about the input and output of the quantum algorithms are much stronger than what is considered in the classical case. Let us reiterate the assumptions: first, we assume that the input data is encoded as amplitudes of a quantum state. While we have such an encoding is possible, preparing such state, in general, requires exponentially many gates. Second, we assumed that any matrices can be embedded into easy to simulate Hamiltonian and easily accessible. This means that the exponential speedup exists ONLY with respect to an oracle that encodes such a Hamiltonian. Lastly, we are only required to output the solution as a quantum state instead of fully writing it out.

The input problem is abstracted by a quantum random memory (QRAM). QRAM can return data as

$$\text{QRAM} \, |j\rangle \, |0 \dots 0\rangle = |j\rangle \, |\psi_j\rangle \tag{37}$$

and can operate in superposition. See https://youtu.be/IicCWK2D7sg for QRAM construction. If the Hermitian matrix encoding the data is low-rang, QRAM allows for efficient implementation of QML algorithms

Surprisingly, Ewin Tang (followed by others) managed to dequantize multiple quantum algorithm that relied on QRAM constructions. Her approach used sampling and randomized algebra techniques. As such, there are currently no known QML algorithms with provable, non-oracular exponential speedup.

3.4 Further Reading

Quantum machine learning
J Biamonte, P Wittek, N Pancotti, P Rebentrost, N Wiebe, S Lloyd
arXiv:1611.09347

A quantum-inspired classical algorithm for recommendation systems
E Tang
arXiv:1807.04271v3

Key questions for the quantum machine learner to ask themselves
N Wiebe
https://iopscience.iop.org/article/10.1088/1367-2630/abac39/meta