

Quantum 101: Do I need a quantum RAM?

SOMETIMES.

WIQCA online seminar

Olivia Di Matteo

Quantum Information Science Associate, TRIUMF

13 May 2020

This discussion

- What does 'quantum machine learning' mean?
- Do I need a quantum RAM to do it?
- If I did, how would I make one?

Machine learning

Mathematical models to learn about patterns in data.

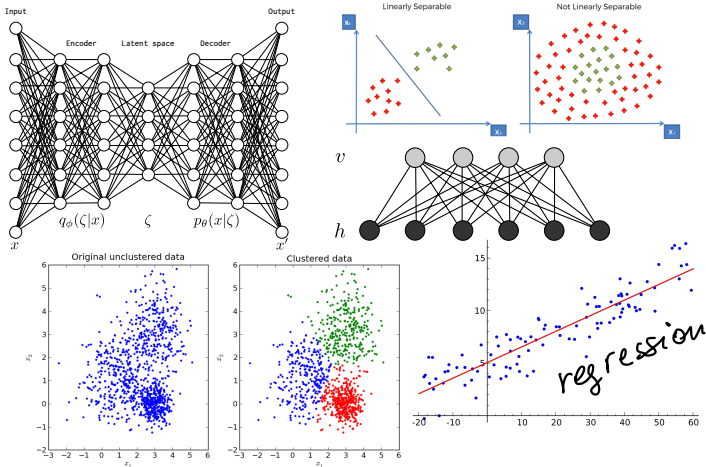
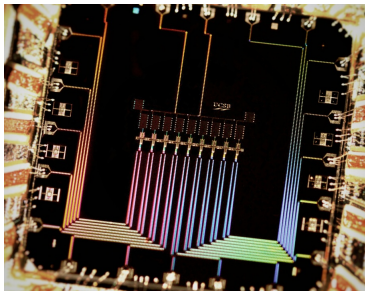


Image credits: <https://towardsdatascience.com/k-means-data-clustering-bce3335d2203>
<https://medium.com/pursuitnotes/day-12-kernel-svm-non-linear-svm-5fdefe77836c>
<https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a>

Quantum computing

A quantum computer manipulates physical quantum systems to solve problems.



Quantum computers will one day perform many currently intractable tasks:

- simulation of physical systems
- optimization
- solving discrete math problems
- ...

Image: <http://web.physics.ucsb.edu/~martinisgroup/> (Photo credit: Julian Kelly).

Quantum machine learning?

There are *known* quantum speedups for a number of problems - is machine learning one of them?

Quantum machine learning?

There are *known* quantum speedups for a number of problems - is machine learning one of them?

Maybe - the field of **quantum machine learning** (QML) is very new!

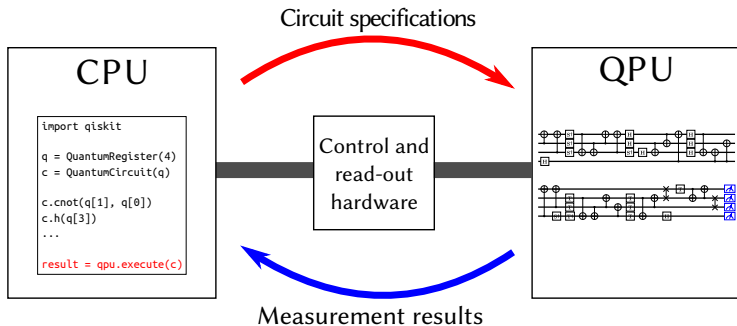
QML can take many forms, and has many definitions:

- Translation of classical models into the language of quantum computers (add a Q to the front of the acronym)
- Outsourcing parts of (or full) ML computations to QC when a speedup is available
- Start from a QC and ask what type of ML will fit its characteristics and constraints (very little done in this area)

Hybrid quantum-classical computing

Most QML algorithms are *hybrid* to some extent:

- use **quantum computers** to help with the ‘hard parts’ of **classical machine learning** and optimization tasks
- use **classical computers** to help with the optimization of parameters in **quantum machine learning**



Quantum machine learning

QML algorithms are often distinguished by the type of data they deal with:

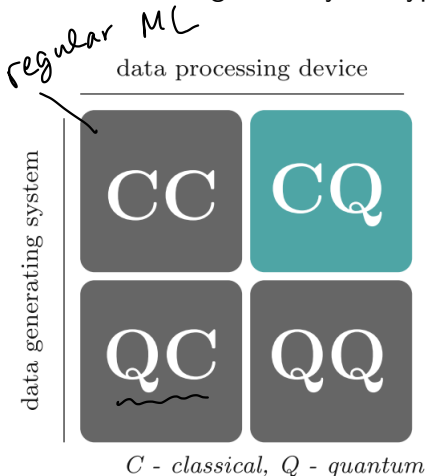


Image credit: M. Schuld and F. Petruccione. Supervised Learning with Quantum Computers. Springer International Publishing, 2018.

QML in this talk

We will focus on the 'CQ' case: ML with **classical data** on a **quantum computer**.

People have designed quantum algorithms for machine learning subroutines that *do* in theory yield a speed up over best known ~~classical methods~~!

Many use the Quantum Linear Systems Algorithm (HHL algorithm) as a subroutine - this gives an exponential speedup¹.

But there's a problem...

¹With conditions on sparsity and condition number of the matrix.

Running a 'CQ' QML algorithm

Usually think of classical data as a bunch of vectors

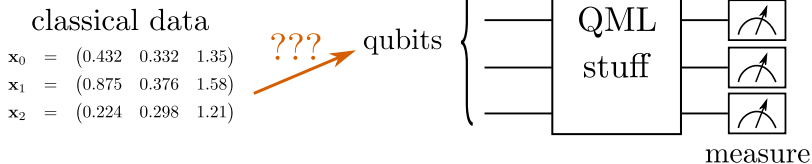
	Feat. 1	Feat. 2	Feat. 3
0	0.432	0.332	1.35
1	0.875	0.376	1.58
2	0.224	0.298	1.21
\vdots	\vdots	\vdots	\vdots

$$\begin{aligned}\Rightarrow \quad \mathbf{x}_0 &= (0.432 \quad 0.332 \quad 1.35) \\ \mathbf{x}_1 &= (0.875 \quad 0.376 \quad 1.58) \\ \mathbf{x}_2 &= (0.224 \quad 0.298 \quad 1.21)\end{aligned}$$

$$X = \text{np.array}([[\dots]])$$

Running a 'CQ' QML algorithm

How do we load classical data on to a quantum computer?



Loading classical data on a quantum computer

Answer: quantum state preparation, or *quantum RAM*

Loading classical data on a quantum computer

Answer: quantum state preparation, or *quantum RAM*

Quantum RAM is hardware and circuitry used to read classical data into a quantum computer for further processing.

Loading classical data on a quantum computer

Answer: quantum state preparation, or *quantum RAM*

Quantum RAM is hardware and circuitry used to read classical data into a quantum computer for further processing.

Problem

*“We made this really cool algorithm. It runs really fast!
...Assuming that I have this very specifically crafted input state and can query a qRAM efficiently.”*

— many recent quantum algorithm papers

Loading classical data on a quantum computer

Answer: quantum state preparation, or *quantum RAM*

Quantum RAM is hardware and circuitry used to read classical data into a quantum computer for further processing.

Problem

*“We made this really cool algorithm. It runs really fast!
...Assuming that I have this very specifically crafted input state and can query a qRAM efficiently.”*

— many recent quantum algorithm papers

It doesn't matter if your QML algorithm yields a speedup over classical ones if it's not efficient to load the data!

Quantum RAM

1. What does it do?
2. How does it work?
3. Is it feasible?
4. Do we really need it?

Qubits

When we load data on a quantum computer, we are creating a qubit state that represents the data.

This is what a qubit 'looks like':

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$$

Here α_0 and α_1 are complex numbers called *amplitudes*, and this state is in a *superposition* of the two 'basis states' $|0\rangle$ and $|1\rangle$.

The state must also be normalized, i.e. $\overbrace{\alpha_0\alpha_0^* + \alpha_1\alpha_1^*}^{\text{prob.}} = 1$

Multi-qubit systems

When we put multiple qubits together, we can make superpositions of *all* possible combinations of the qubits being in their basis states:

$$|\psi\rangle = \alpha_0|000\rangle + \alpha_1|001\rangle + \alpha_2|010\rangle + \dots$$



If we have n qubits, there are 2^n available amplitudes.

Multi-qubit systems

When we put multiple qubits together, we can make superpositions of *all* possible combinations of the qubits being in their basis states:

$$|\psi\rangle = \alpha_0|000\rangle + \alpha_1 \underbrace{|001\rangle} + \alpha_2|010\rangle + \dots$$

If we have n qubits, there are 2^n available amplitudes.

Both the *amplitude* and the *basis state* can carry information about classical data.

Amplitude encoding

One way to load classical data on a quantum computer is to *encode* the classical data into the amplitudes - this is called an *amplitude encoding*.

Mathematically,

$$\mathbf{x} = (x_1 \cdots x_n) \rightarrow \sum_i \underbrace{\frac{x_i}{\|\mathbf{x}\|}} |i\rangle$$

Example

Suppose

$$\mathbf{x} = (0.212 \quad 0.123 \quad 0.012 \quad 0.543) \leftarrow$$

We can encode this into a 2-qubit state:

$$|\psi\rangle = 0.356|00\rangle + 0.206|01\rangle + 0.020|10\rangle + 0.911|11\rangle$$

Basis encoding

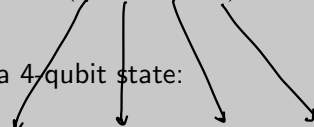
Turn data into binary strings and create superpositions of those basis states.

Example

Suppose

$$\mathbf{x} = (3 \quad 4 \quad 7 \quad 10)$$

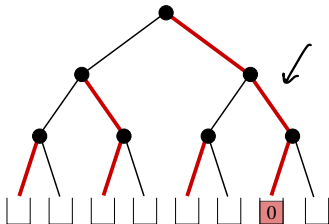
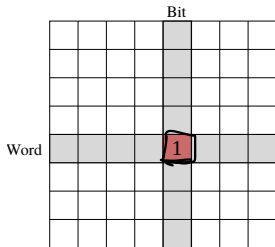
We can encode this in a 4-qubit state:

$$|\psi\rangle = \frac{1}{2} (|0011\rangle + |0100\rangle + |0111\rangle + |1010\rangle)$$


Quantum random access memory

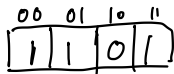
Quantum RAM is a type of encoding that loads binary classical data *in superposition*.

Classical RAM is an indispensable resource in classical computers - store the active state of a computation, load in data, etc.



Memory cells typically consist of transistor / capacitor pair. Cells are *individually addressable* by activating specific combinations of transistors.

Question 1: What does it do?



Suppose we have n -bit memory addresses $\mathbf{a} = a_0 \cdots a_{n-1}$.

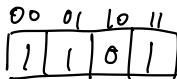
Each address contains a 0 or a 1, denoted by $b_{\mathbf{a}}$.

In a classical RAM, each time we query the RAM we ask “what are the contents of the RAM at address \mathbf{a} , or

$$\mathbf{a} = a_0 \cdots a_{n-1} \xRightarrow{\text{RAM}} b_{\mathbf{a}}$$

$$\text{RAM}(01) \rightarrow 1$$

Question 1: What does it do?



In a quantum RAM, we can query *multiple locations* by querying in *superposition*!

$$\sum_i |a_i\rangle |0\rangle \xrightarrow{\text{qRAM}} \sum_i |a_i\rangle |b_{a_i}\rangle$$

Some algorithms intrinsically require data to be loaded in this form; others use it as a subroutine to perform amplitude encoding.

$$(|00\rangle + |01\rangle + i|10\rangle + |11\rangle) |0\rangle$$

$$\xrightarrow{\text{qRAM}} |00\rangle |1\rangle + |01\rangle |1\rangle + |10\rangle |0\rangle + |11\rangle |1\rangle$$

Question 2: How does it work?



Method 1: 'explicit' qRAM

Store data in actual, physical qubits set to $|0\rangle$ or $|1\rangle$, then design a quantum circuit that will extract the data.

Advantage:

- Easy to change the contents of the memory.

Disadvantage:

- Need a lot of extra qubits for storage space!

Question 2: How does it work?

Method 2: 'implicit' qRAM

Design a quantum circuit that simply gives the desired output for each address.

Advantage:

- No 'data qubits' needed to store data.

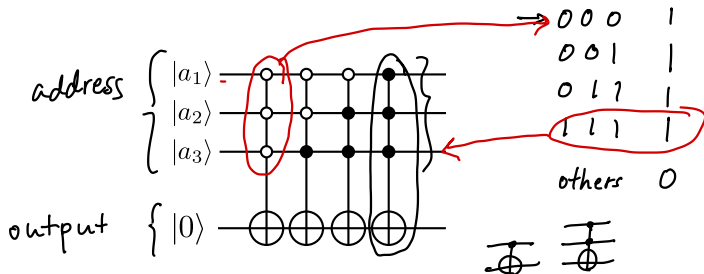
Disadvantage:

- Need to know contents of memory in advance
- Hard to change - need to rewrite and reoptimize the circuit.

This type of qRAM is more like a *lookup table*, or *qROM*.

Question 2: How does it work?

An example of an implicit qRAM: the circuit in the talk ad!

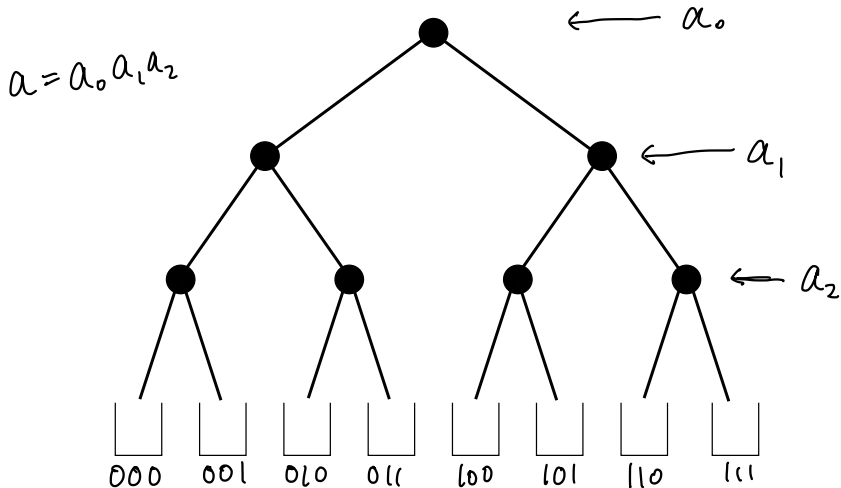


Feed circuit an address - circuit changes an output bit to $|1\rangle$ if there is a 1 at that address.

Each part of the circuit does this for a *different* address - need to know the contents of your memory in advance!

Question 2: How does it work?

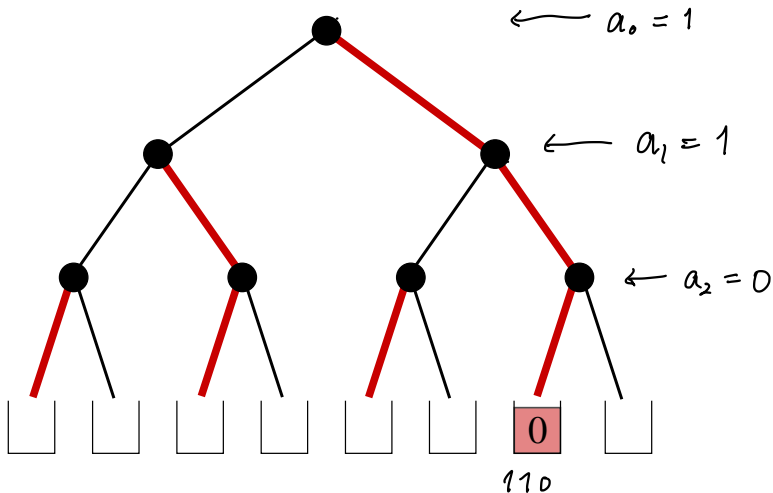
Let's look at the original proposal for qRAM - data stored in leaves of a binary tree. With n address bits we can query 2^n memory cells.



Fanout RAM

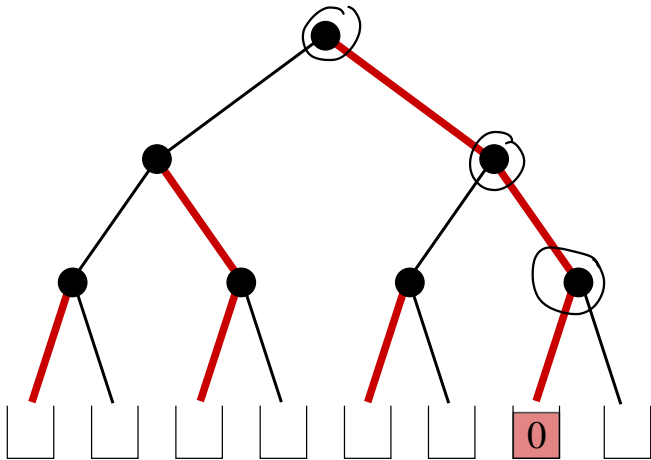
The k^{th} address bit controls all transistors in row k .

Example: query memory location 110.



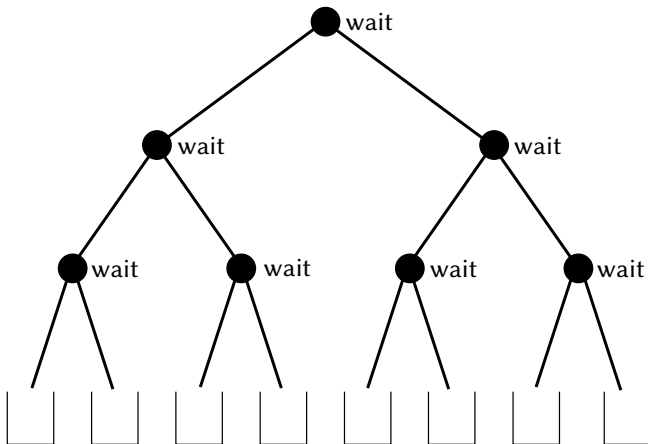
Fanout RAM

Problem: need to activate $2^n - 1$ nodes, but only n of them are actually used to get us to the address we want.



Improved version: bucket brigade RAM

Each node of the tree is a *trit* that tells us which direction to take. Trit states are wait, left, and right. All trits start in **wait**.

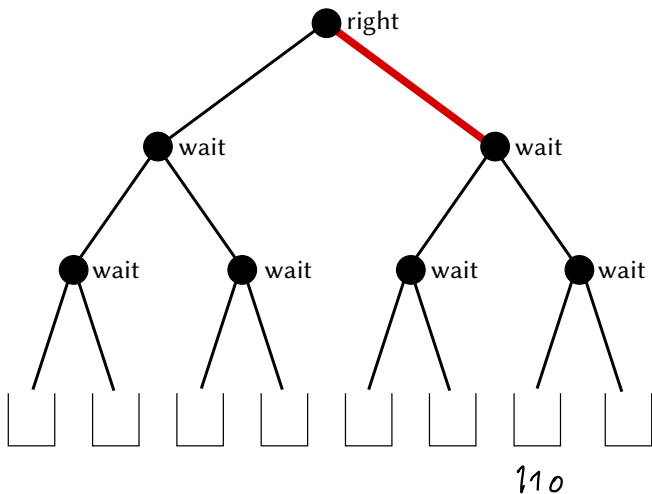


V. Giovannetti et al. *Quantum random access memory* (2008) Phys. Rev. Lett. **100**, 160501

V. Giovannetti et al. *Architectures for a quantum random access memory* (2008) Phys. Rev. A **78**, 052310

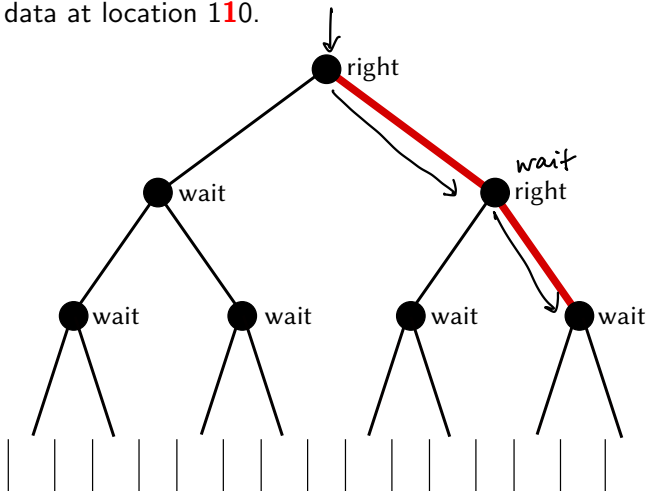
Improved version: bucket brigade RAM

Query data at location **110**.



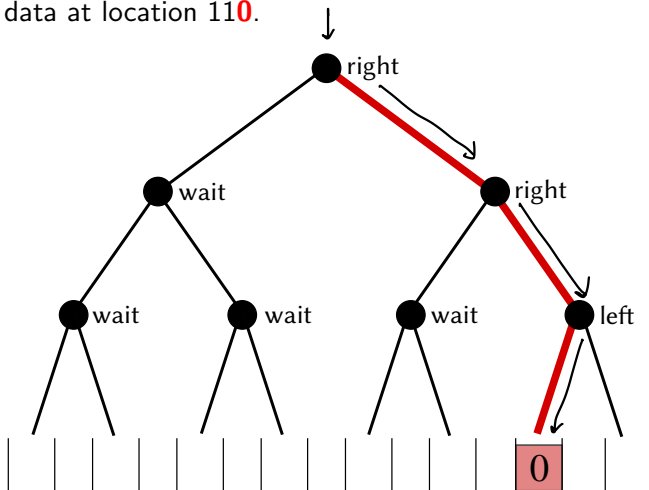
Improved version: bucket brigade RAM

Query data at location 110.



Improved version: bucket brigade RAM

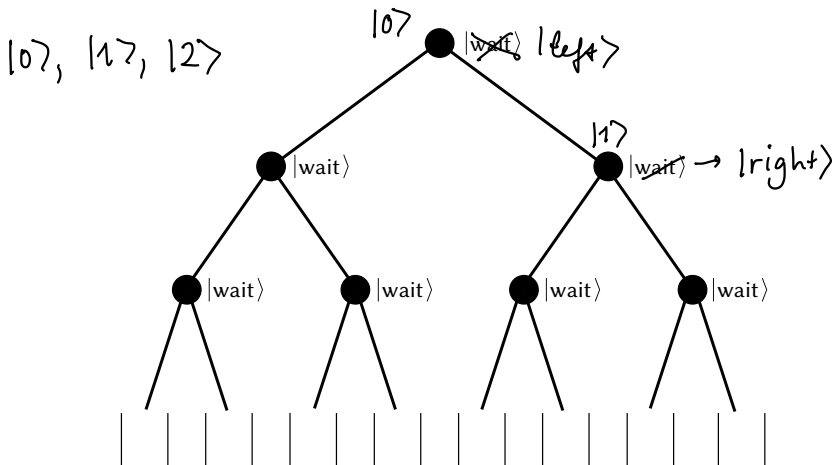
Query data at location 11**0**.



Bucket brigade qRAM

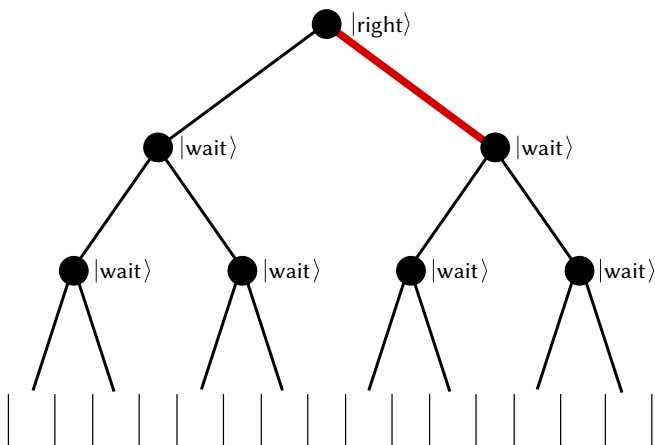
Trits become *qutrits*. Design reversible transformations that turn:

$$|0\rangle|wait\rangle \rightarrow |stuff\rangle|left\rangle, \quad |1\rangle|wait\rangle \rightarrow |stuff\rangle|right\rangle \quad (1)$$



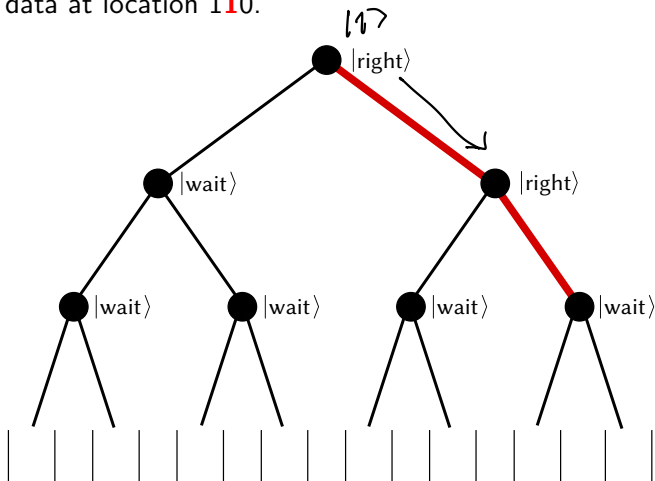
Bucket brigade qRAM

Query data at location **110**. $|1\rangle$



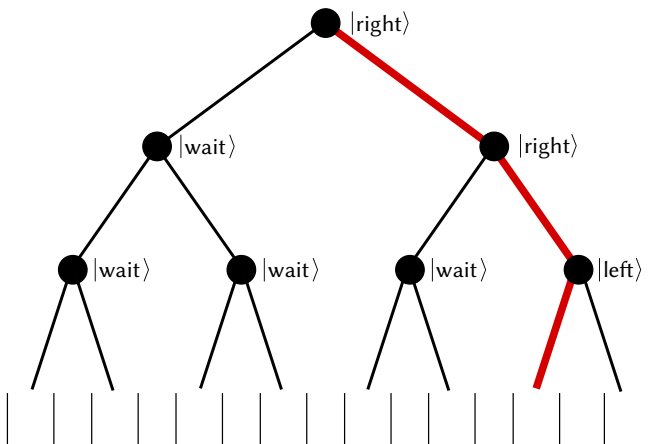
Bucket brigade qRAM

Query data at location 110.



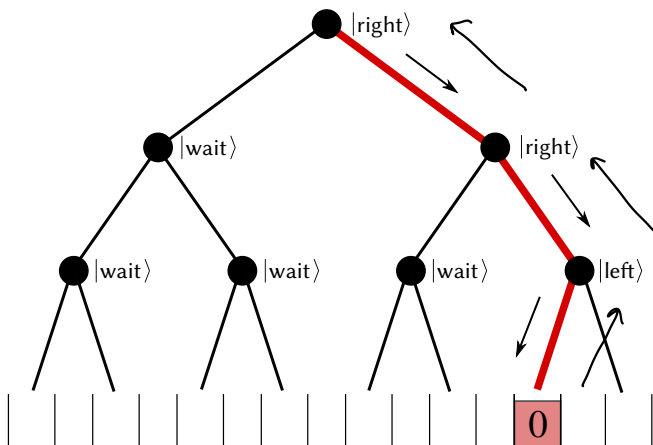
Bucket brigade qRAM

Query data at location 110.



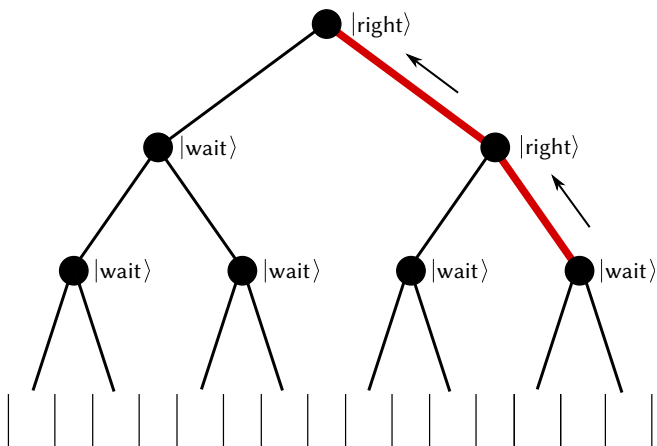
Bucket brigade qRAM

Send a **bus qubit** through to couple to the memory cell.



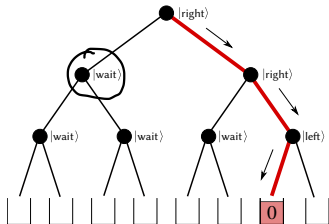
Bucket brigade qRAM

Send it back the way it came and reset qutrits to $|wait\rangle$.



Question 3: Is it feasible in the future?

Many advantages to bucket brigade qRAM...



- uses a *polynomial* number of operations. $O(n^2)$
- many nodes are idle

This suggests that:

- can get away with error rates that are only polynomially small
- things that are idle don't need to be under active error correction

Question 3: Is it feasible in the future?

But is this really the case??

- Unclear (to me) how that would work if you're querying in superposition - you're lighting up all the nodes!
- Later work has proven that when you have to do exponentially many qRAM queries (e.g. Grover's algorithm), you need to have exponentially small error rates which means the whole qRAM must be actively error corrected anyways

Question 3: Is it feasible in the future?

But is this really the case??

- Unclear (to me) how that would work if you're querying in superposition - you're lighting up all the nodes!
- Later work has proven that when you have to do exponentially many qRAM queries (e.g. Grover's algorithm), you need to have exponentially small error rates which means the whole qRAM must be actively error corrected anyways

But - for cases with a *small number of queries*, we might be able to get away with this...

Question 3: Is it feasible in the future?

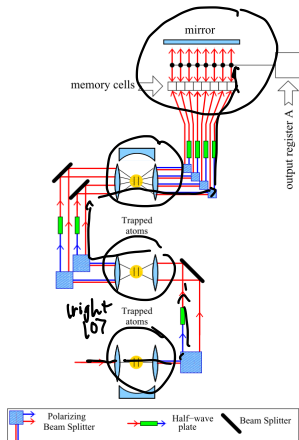
But is this really the case??

- Unclear (to me) how that would work if you're querying in superposition - you're lighting up all the nodes!
- Later work has proven that when you have to do exponentially many qRAM queries (e.g. Grover's algorithm), you need to have exponentially small error rates which means the whole qRAM must be actively error corrected anyways

But - for cases with a *small number of queries*, we might be able to get away with this...

... assuming we could build one.

Question 3: Is it feasible in the future?



No one has done so, to the best of my knowledge, but there have been some proposals.

- photonic²
- phononic³
- atoms in cavities

We still need 2^n storage qubits to maintain coherence through the entire computation - hard when coherence times are low!

²V. Giovannetti et al. *Architectures for a quantum random access memory* (2008) Phys. Rev. A **78**, 052310

³Pretty recent: <https://arxiv.org/abs/1906.11340>

Question 3: Is it feasible in the future?

What about 'implicit' qRAM, or qROM, that are based only on 'normal' quantum circuits?

These have their own issues...

- Complexity of circuits is high
- Circuits need to be compiled and optimized, and re-optimized if memory changes
- Still have the problem of decoherence - need to implement *fault-tolerantly*, which causes a massive overhead in the number of additional qubits and time required to perform error correction

Question 4: Do we really need it?

Answer: **Sometimes.**

Many 'CQ' QML algorithms *do* need a qRAM, when they have to load classical data.

But many others *don't*, like 'QQ' algorithms, where data comes directly from a quantum process.

And even some 'CQ' algorithms might not need it, but for different reasons...

Question 4: Do we really need it?

In some cases, maybe don't need it at all if we can 'dequantize' the QML algorithm!

arXiv.org > quant-ph > arXiv:1603.08675

Help | Advan

Quantum Physics

[Submitted on 29 Mar 2016 (v1), last revised 22 Sep 2016 (this version, v3)]

Quantum Recommendation Systems

Iordanis Kerenidis, Anupam Prakash

A recommendation system uses the past purchases or ratings of n products by a group of m users, in order to provide personalized recommendations to individual users. The information is modeled as an $m \times n$ preference matrix which is assumed to have a good rank- k approximation, for a small constant k . In this work, we present a quantum algorithm for recommendation systems that has running time $O(\text{poly}(k)\text{polylog}(mn))$. All known classical algorithms for recommendation systems that work through reconstructing an approximation of the preference matrix run in time polynomial in the matrix dimension. Our algorithm provides good recommendations by sampling efficiently from an approximation of the preference matrix, without reconstructing the entire matrix. For this, we design an efficient quantum procedure to project a given vector onto the row space of a given matrix. This is the first algorithm for recommendation systems that runs in time polylogarithmic in the dimensions of the matrix and provides an example of a quantum machine learning algorithm for a real world application.

Question 4: Do we really need it?

In some cases, maybe don't need it at all if we can 'dequantize' the QML algorithm!

arXiv.org > cs > arXiv:1807.04271

search...

Help | Advanced Search

Computer Science > Information Retrieval

[Submitted on 10 Jul 2018 (v1), last revised 9 May 2019 (this version, v3)]

A quantum-inspired classical algorithm for recommendation systems

Ewin Tang

We give a classical analogue to Kerenidis and Prakash's quantum recommendation system, previously believed to be one of the strongest candidates for provably exponential speedups in quantum machine learning. Our main result is an algorithm that, given an $m \times n$ matrix in a data structure supporting certain ℓ^2 -norm sampling operations, outputs an ℓ^2 -norm sample from a rank- k approximation of that matrix in time $O(\text{poly}(k) \log(mn))$, only polynomially slower than the quantum algorithm. As a consequence, Kerenidis and Prakash's algorithm does not in fact give an exponential speedup over classical algorithms. Further, under strong input assumptions, the classical recommendation system resulting from our algorithm produces recommendations exponentially faster than previous classical systems, which run in time linear in m and n .


Summary and outlook: QML

- QML refers primarily to two cases:
 - processing classical data on quantum computers
 - processing quantum data on quantum computers
- Many QML algorithms require exchange between classical and quantum computers
- Some QML algorithms may yield a speedup, with access to an efficient qRAM (and that they don't later get 'dequantized')
- QML is a very new field, with lots to be discovered - hard to design new algorithms, but could be very rewarding

Summary and outlook: qRAM

- *Some* QML algorithms require a qRAM to load classical data
- qRAMs exist in the sense that we can write down circuits and experimental proposals, but...
 - they are probably very hard to build
 - the quantum circuits may not necessarily be efficient
 - if we need to do a large number of queries with full error correction, the overhead might negate any speedups from our QML algorithm

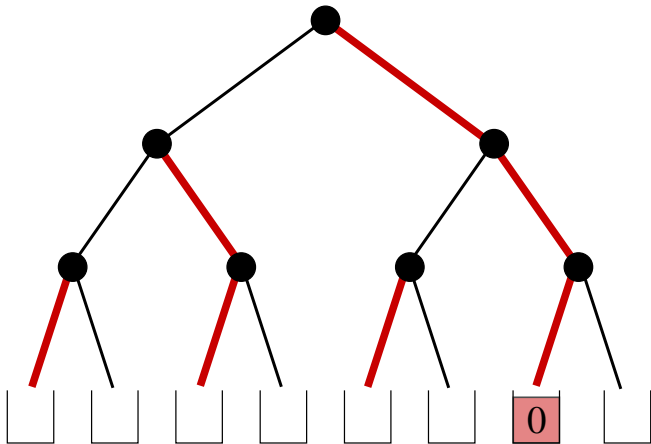
This is an active area of research!



In PAC learning, every function that can be learned with a polynomial number of quantum queries can also be learned with a polynomial number of classical queries (Servedio and Gortler). But, this equivalence does not hold for computing time.

Fanout qRAM?

Bigger problem: Would need to couple the k^{th} address qubit with the 2^k qubits in the k^{th} row. Very delicate superposition, would need error rates on order of $O(2^{-n})$.



Basis encoding: linear in $O(MN)$ where N number of data points and M number of features; need to do for each feature.

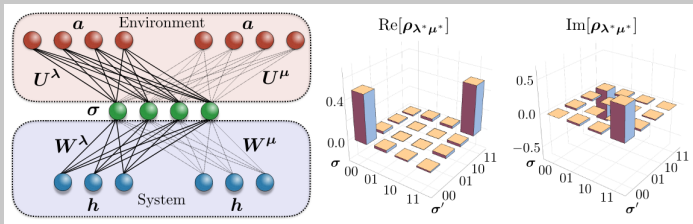
Quantum machine learning

CC	CQ
QC	QQ

Processing data generated by **quantum systems** using **classical machine learning**.

Example: quantum tomography

Use machine learning to determine the state of a quantum system based on measurements.



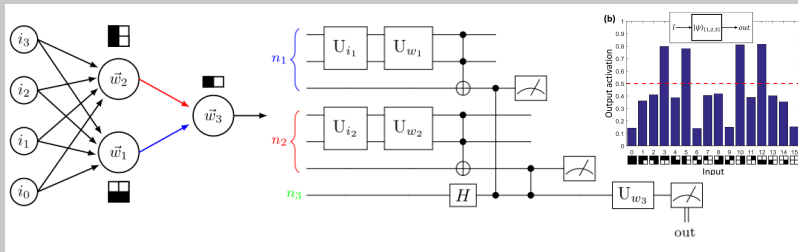
Quantum machine learning

CC	CQ
QC	QQ

Doing machine learning with **classical data** on **quantum computer**.

Example: quantum neural network implementation

Classifies 4-bit images. Actually implemented!



Quantum machine learning

CC	CQ
QC	QQ

Doing machine learning with **quantum data** on a **quantum computer**.

Example: classifying phases of matter

Learn parameters of a quantum convolutional neural network - measurement result tells us the phases of the input state.

