Quantum Computing

Python Deep Dive

Week 2: Quantum Algorithms: A brief Discussion

Lecture Flow

Quick recap

Shor’s Algorithm

Variational Quantum Regression

Hybrid Quantum - Classical Neural Network (if time permits)

Quick Recap

Qubits

- 1 unit of information in quantum computers.
- Exists as a superposition of two values.
- Superposition: a qubit is a linear combination of $|0\rangle$ and $|1\rangle$
- Measuring the qubits 'collapses' the superposition.

- This is how we represent qubits: (symbol of psi)

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Quantum Gates

- Maps one the quantum state to another quantum state
- Moves a point on the Bloch Sphere to another point
- We can represent them with matrices

Some basic Quantum gate matrices

Pauli gates:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$ Pauli-X operator (X)

$$\sigma_y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}$$ Pauli-Y operator (Y)

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$ Pauli-Z operator (Z)

Hadamard gate

The Hadamard gate is a single-qubit operation that maps the basis state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, thus creating an equal superposition of the two basis states.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

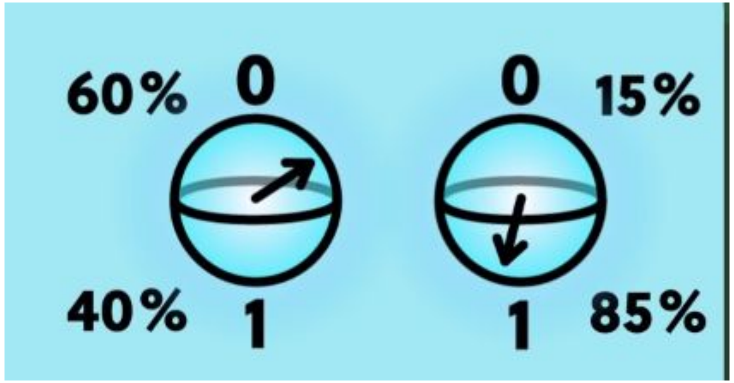
1. **Superposition Principle**

   (Qubits exist as superpositions of two states.)

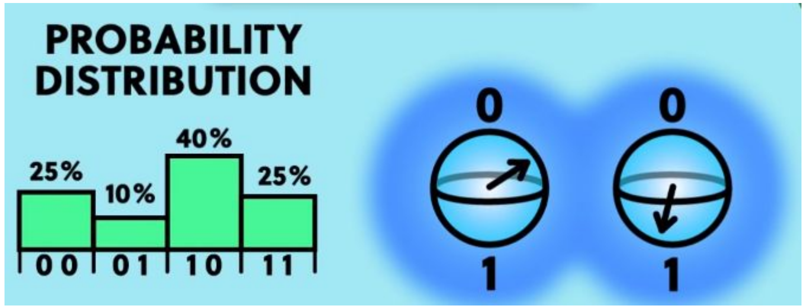
2. **Quantum Entanglement**
3. **Interference**

Quantum Entanglement

- In classical computers, each bit is independent of other bits.
- Qubits can be 'entangled', which means they can be made dependent on each other.
- We can no longer think of entangled qubits as two individual qubits, but as a system of qubits.



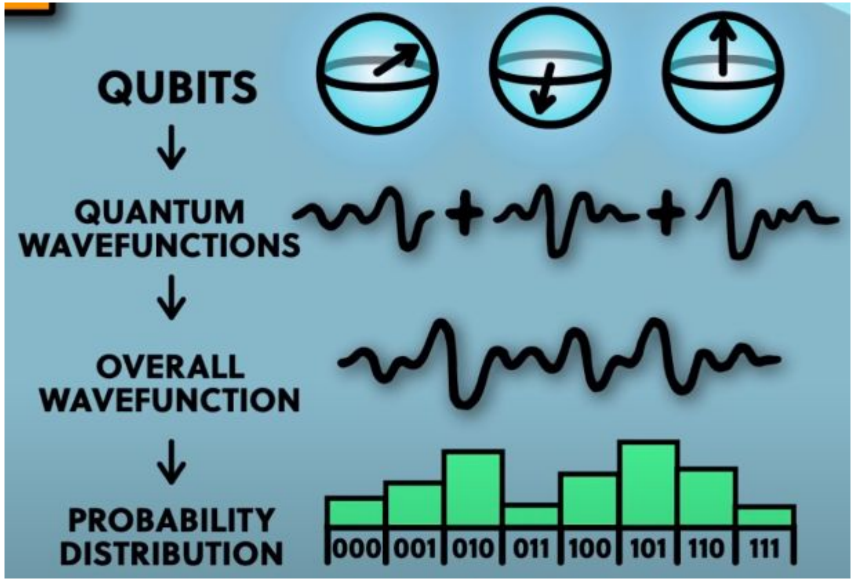
2 qubits



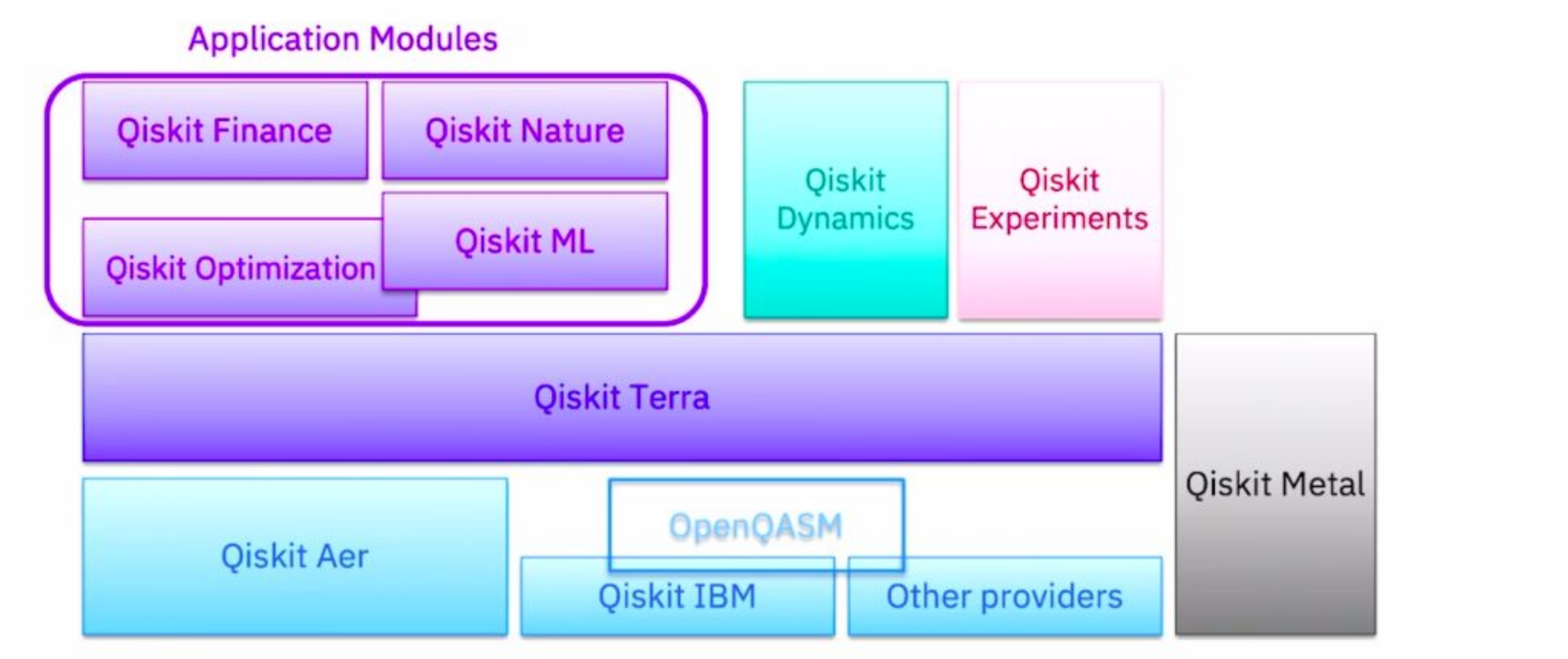
2 entangled qubits

Interference

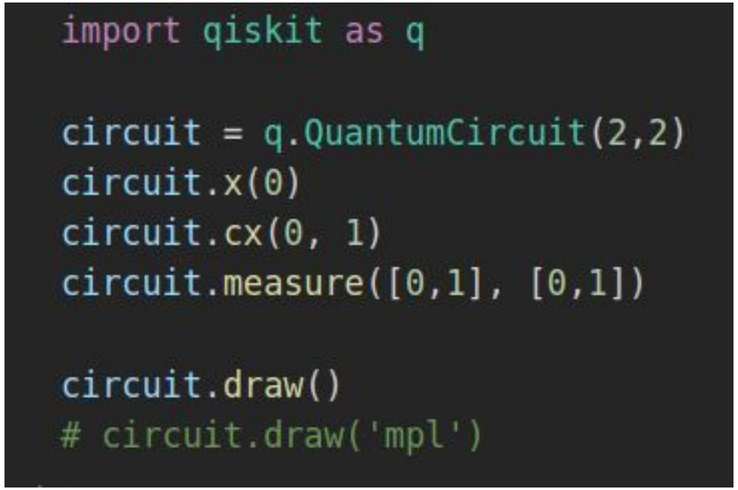
- Particles at the atomic level are represented by wavefunctions.
- When these particles interact, their wavefunctions are added. This is called interference.
- In quantum computers, interference is used to implement gates and measure qubits, at the hardware level.



Structure of Qiskit



Making a basic circuit with qiskit



```
import qiskit as q

circuit = q.QuantumCircuit(2,2)
circuit.x(0)
circuit.cx(0, 1)
circuit.measure([0,1], [0,1])

circuit.draw()
# circuit.draw('mpl')
```

Shor’s Algorithm

Prime Number Multiplication

Problem:

Given a product of two large prime numbers, **n**, find its factors.

Algorithm

1. Make a random guess **g**.

2. Generate a number **p** using quantum computing

3. $\mathbf{g^{p/2} + 1}$ is a better guess.

Finding p

Problem:

Find a number **p** such that $\mathbf{g^{p/2} + 1}$ has higher chances of being a factor of **n**.

Fact 1

If **A** and **B** are co-primes, then there exists a **p** such that
$\mathbf{A^p = m.B + 1}$

Finding p

$g^p = m.B + 1$

$g^p - 1 = m.B + 1$

$(g^{p/2}+1).(g^{p/2}-1) = m.B$

We might have a common factor here.

Solving the subproblem

Problem

For any two coprime numbers **A** and **B** , find positive p such that

$$A^p = m.B + 1$$

Algorithm

Send a superposition of all **p** from **1 to A** to the exponentiation function.

Destructively interfere all answers that don’t give remainder 1.

Solving the subproblem

Fact:

If p gives remainder 1, then for any other number x:

$$g^{x} = m \cdot N + r$$

$$\Downarrow$$

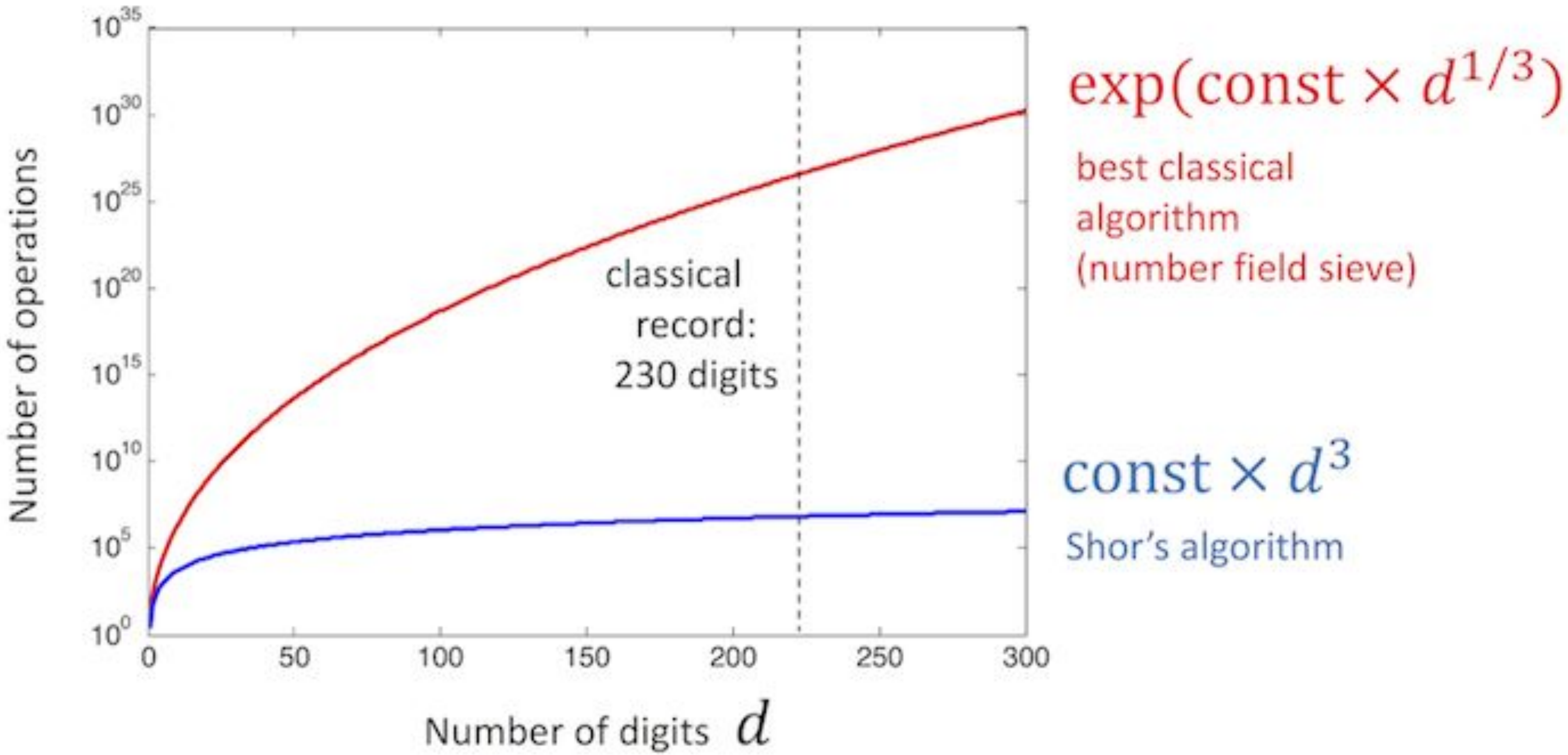
$$g^{x+p} = m_2 \cdot N + r$$

Algorithm

1. Measure just the remainder
2. All other number are periodic with period p
3. Find frequency using Fourier Transform
4. p = 1/frequency

Solving the subproblem

$$\underbrace{g^{x} \qquad g^{x+p} \qquad g^{x-p} \qquad g^{x+2p}}_{+r}$$


Number of operations
10^{0}
10^{5}
10^{10}
10^{15}
10^{20}
10^{25}
10^{30}
10^{35}
0
50
100
150
200
250
300
Number of digits d
classical record: 230 digits
$\exp(\text{const} \times d^{1/3})$
best classical algorithm (number field sieve)
$\text{const} \times d^{3}$
Shor's algorithm

Linear Regression Review

Let’s jump onto this https://mlu-explain.github.io/linear-regression/.

Variational Quantum Regression

Introduction

- Here we create a protocol for linear regression which can exploit the properties of a quantum computer. Our dataset consists of N 2D data points. They follow the equation of the form:

$$y = ax + b$$

- Let’s first theoretically explore this proposed algorithm, and then we will be looking at the code for the same.
- Please note that as we discussed that we cannot just use a quantum computer for any arbitrary tasks since quantum computers are not here to replace the classical computers but to improve on their shortcomings.
- So, this algorithm has no known advantage over the most widely-used classical algorithm (Least Squares Method), but does nicely demonstrate the different elements of variational quantum algorithms.

Let’s hold up!

Let’s just recap, what is an ansatz?

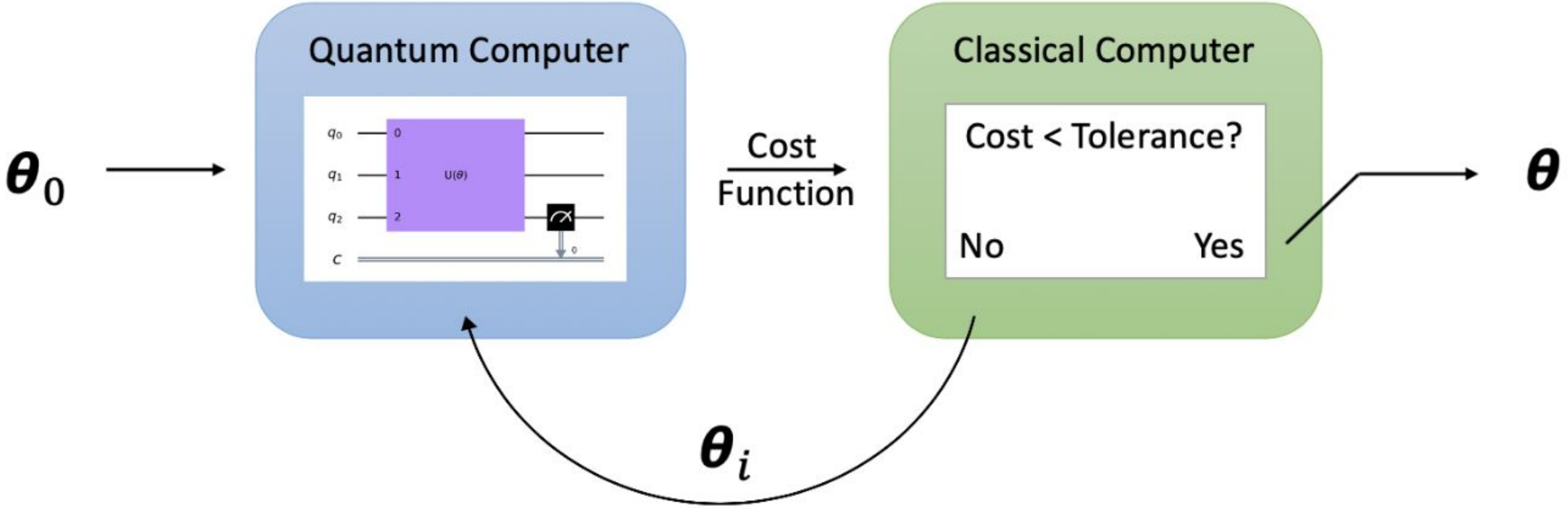
> An ansatz is an assumption about the form of an unknown function which is made in order to facilitate solution of an equation or other problem.

In variational quantum computing, we can say that we propose an initial solution to a problem, an ansatz.

Variational Quantum Computing

Variational quantum computing exploits the advantages of both classical computing and quantum computing.

- We propose an initial solution to a problem, an ansatz.
- In our case our ansatz will be an ansatz parametrised by a and b.
- We then prepare our qubits and test how good the ansatz is, using the quantum computer.
- Testing the ansatz equates to minimising a cost function. We feed the result of this cost function back to the classical computer, and use some classical optimisers to improve on our ansatz, i.e. our initial guesses for a and b.
- We repeat this process until the ansatz is good enough within some tolerance.


$\boldsymbol{\theta}_0$
Quantum Computer
q_0
q_1
q_2
c
0
1
2
U(θ)
0
Cost Function
Classical Computer
Cost < Tolerance?
No
Yes
$\boldsymbol{\theta}$
$\boldsymbol{\theta}_i$

Translate to Quantum Domain

Let us now look at encoding the dataset onto the quantum bits.

Let us think of y as a length N vector. The easiest way to encode this data set onto a quantum computer is by initialising qubits in the state $|y\rangle$,

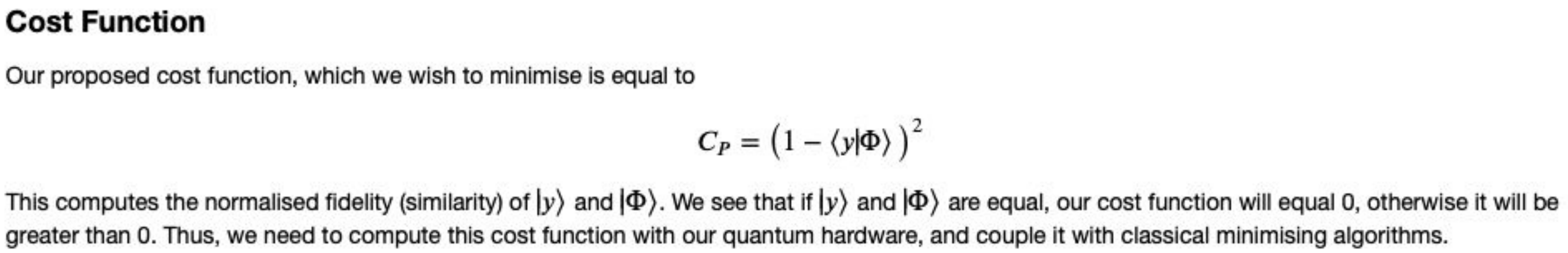
where $|y\rangle = \frac{1}{C_y}\vec{y}$ and C_y is a normalisation factor.

Now we propose a trial solution, or ansatz, which is parametrized by a and b, as follows:

$$|\Phi\rangle = \frac{1}{C_\Phi}(a\vec{x} + b)$$

where C_Φ is again a normalisation factor.

Due to the definition of the tensor product and the fact that the general statevector of a single qubit is a vector of length 2, n qubits can encode length-2^n vectors.



Cost Function

Our proposed cost function, which we wish to minimise is equal to

$$C_P = \left(1 - \langle y|\Phi\rangle\right)^2$$

This computes the normalised fidelity (similarity) of $|y\rangle$ and $|\Phi\rangle$. We see that if $|y\rangle$ and $|\Phi\rangle$ are equal, our cost function will equal 0, otherwise it will be greater than 0. Thus, we need to compute this cost function with our quantum hardware, and couple it with classical minimising algorithms.

On looking at the inner product part of this cost function we realise that overall cost function is just centric to the similarity between the true and predicted states.

A quick look at tensor and inner products.....

Inner Product

We use the inner product to find the overlap between two quantum states

braket (bra+ket): $\langle\psi|\varphi\rangle$

- $\langle 0|0\rangle$: $\begin{pmatrix}1 & 0\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} = 1\times1 + 0\times0 = 1$
- $\langle 1|0\rangle$: $\begin{pmatrix}0 & 1\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} = 1\times0 + 0\times1 = 0$
- $\langle 0|1\rangle$: $\begin{pmatrix}1 & 0\end{pmatrix}\cdot\begin{pmatrix}0\\1\end{pmatrix} = 1\times0 + 0\times1 = 0$
- $\langle 1|1\rangle$: $\begin{pmatrix}0 & 1\end{pmatrix}\cdot\begin{pmatrix}0\\1\end{pmatrix} = 0\times0 + 1\times1 = 1$

E.g.

$$x(y+z) = x\cdot y + x\cdot z$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$\langle 0|\psi\rangle = \langle 0|\left(\alpha|0\rangle + \beta|1\rangle\right) = \alpha\langle 0|0\rangle + \beta\langle 0|1\rangle$$

$$= \alpha + \beta\cdot 0 = \alpha$$

$$|\psi\rangle = \tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle$$

$$|\phi\rangle = \tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle$$

$$\langle\psi|\phi\rangle = \left(1/\sqrt{2} \quad 1/\sqrt{2}\right)\cdot\begin{pmatrix}1/\sqrt{2}\\ -1/\sqrt{2}\end{pmatrix} = 1/2 - 1/2 = 0$$

$$\hookrightarrow \left(1/\sqrt{2}\,\langle 0| + 1/\sqrt{2}\,\langle 1|\right)\cdot\left(1/\sqrt{2}\,|0\rangle - 1/\sqrt{2}\,|1\rangle\right)$$

$$1/2\,\overset{1}{\cancel{\langle 0|0\rangle}} + 1/2\,\overset{0}{\cancel{\langle 1|0\rangle}} - 1/2\,\overset{0}{\cancel{\langle 0|1\rangle}} - 1/2\,\overset{1}{\cancel{\langle 1|1\rangle}}$$

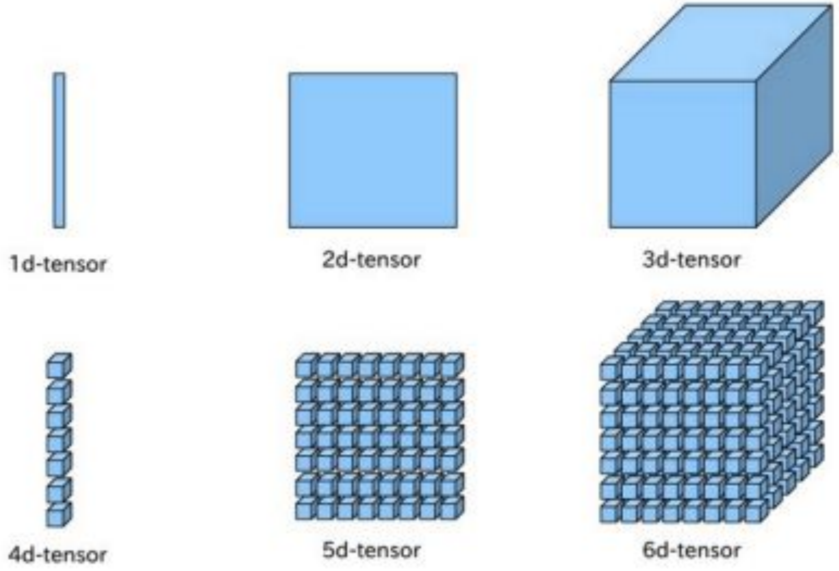
$$= 1/2 + 1/\cancel{2\times} 0 - 1/\cancel{2\times} 0 - 1/2 = 0$$

Two states |ψ⟩ and |φ⟩ are ***"orthogonal"*** if: ⟨ψ | φ⟩=0 (orthogonal=perpendicular)

$|0\rangle$ and $|1\rangle$ are orthogonal $\langle 0|1\rangle = 0$

Tensor

A tensor is essentially an n-dimensional vector… In quantum computing, we care about vectors (states) and matrices (gates), which are 1-D and 2-D tensors!



$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}, \qquad B = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix},$$

respectively, then the tensor product of these two matrices is

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2} \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}.$$

Write out the full matrix for the following tensor products…

(1) $\begin{pmatrix}1\\2\end{pmatrix}\otimes\begin{pmatrix}3\\4\end{pmatrix}=$

(3) $\begin{pmatrix}1\\-1\end{pmatrix}\otimes\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}=$

(2) $\begin{pmatrix}3\\4\end{pmatrix}\otimes\begin{pmatrix}1\\2\end{pmatrix}=$

(4) $\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}\otimes\begin{pmatrix}1\\-1\end{pmatrix}=$

Write out the full matrix for the following tensor products...

(1)
$$\begin{pmatrix}1\\2\end{pmatrix}\otimes\begin{pmatrix}3\\4\end{pmatrix} = \begin{pmatrix}1\begin{pmatrix}3\\4\end{pmatrix}\\[4pt] 2\begin{pmatrix}3\\4\end{pmatrix}\end{pmatrix} = \begin{pmatrix}3\\4\\6\\8\end{pmatrix}$$

(2)
$$\begin{pmatrix}3\\4\end{pmatrix}\otimes\begin{pmatrix}1\\2\end{pmatrix} = \begin{pmatrix}3\begin{pmatrix}1\\2\end{pmatrix}\\[4pt] 4\begin{pmatrix}1\\2\end{pmatrix}\end{pmatrix} = \begin{pmatrix}3\\6\\4\\8\end{pmatrix}$$

(3)
$$\begin{pmatrix}1\\-1\end{pmatrix}\otimes\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix} = \begin{pmatrix}1\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}\\[4pt] -1\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}\end{pmatrix} = \begin{pmatrix}2 & 3\\4 & 5\\-2 & -3\\-4 & -5\end{pmatrix}$$

$$\neq$$

(4)
$$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}\otimes\begin{pmatrix}1\\-1\end{pmatrix} = \begin{pmatrix}2\begin{pmatrix}1\\-1\end{pmatrix} & 3\begin{pmatrix}1\\-1\end{pmatrix}\\[4pt] 4\begin{pmatrix}1\\-1\end{pmatrix} & 5\begin{pmatrix}1\\-1\end{pmatrix}\end{pmatrix} = \begin{pmatrix}2 & 3\\-2 & -3\\4 & 5\\-4 & -5\end{pmatrix}$$

Getting back to the topic at hand.....

Computing Inner Products on a Quantum Computer

It is clear we now need a quantum algorithm for computing inner products. Let us go through the theory of computing the inner product $\langle x|y\rangle$ here, which will be translated to quantum hardware in a couple of sections.

Firstly, assume we have a state: $|\phi\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle |x\rangle + |1\rangle |y\rangle \big)$

where we want to find the inner product, ⟨x|y⟩. Applying a Hadamard gate on the first qubit,

$$|\tilde{\phi}\rangle = \frac{1}{2}\Big(|0\rangle \big(|x\rangle + |y\rangle \big) + |1\rangle \big(|x\rangle - |y\rangle \big)\Big)$$

This means that the probability to measure the first qubit as |0⟩ in the computational basis equals:

$$P(0) = \frac{1}{2}\Big(1 + Re\big[\langle x|y\rangle \big]\Big)$$

This follows because:

$$\begin{aligned} P(0) &= \left| \langle 0| \otimes \mathbb{1} \, |\tilde{\phi}\rangle \right|^2 \\ &= \frac{1}{4} \left| |x\rangle + |y\rangle \right|^2 \\ &= \frac{1}{4} \left(\langle x|x\rangle + \langle x|y\rangle + \langle y|x\rangle + \langle y|y\rangle \right) \\ &= \frac{1}{4} \left(2 + 2Re\left[\langle x|y\rangle \right] \right) \\ &= \frac{1}{2} \left(1 + Re\left[\langle x|y\rangle \right] \right) \end{aligned}$$

After a simple rearrangement, we see that

$$Re\left[\langle x|y\rangle \right] = 2P(0) - 1$$

It follows from a similar logic that if we apply a phase rotation on our initial state:

$$|\phi\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle \, |x\rangle - i \, |1\rangle \, |y\rangle \right)$$

then the probability of the same measurement:

$$P(0) = \frac{1}{2} \left(1 + Im\left[\langle x|y\rangle \right] \right)$$

We can then combine both probabilities to find the true ⟨x|y⟩. For this work, we assume that our states are fully real, and so just need the first measurement.

- It should be noted here that qiskit orders its qubits with the last qubit corresponding to the left of the tensor product.
- For the example run, we will compute the inner product of length-8 vectors. Thus, we require 4 qubits (8+8=16=2^4) to encode the state:

$$
\begin{aligned}
|\phi\rangle &= \frac{1}{\sqrt{2}}(|0\rangle|x\rangle + |1\rangle|y\rangle) \\
&= \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}y_1\\y_2\\\vdots\\y_n\end{bmatrix}\right) \\
&= \frac{1}{\sqrt{2}}\left(\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\\y_1\\y_2\\\vdots\\y_n\end{bmatrix}\right)
\end{aligned}
$$

- Finally, in order to measure the probability of measuring the bottom (leftmost) qubit as $|0\rangle$ in the computational basis, we can find the exact theoretical value by finding the resultant statevector and summing up the amplitude squared of the first 2^{n-1} entries (i.e. half of them).

- On a real quantum computer, we would just have to perform the actual measurement many times over, and compute the probability that way.

- Let us now jump into the implementation starting with the code for the theoretical approach and then the approach for actual Quantum Computer.

Hybrid Quantum - Classical Neural Network

Let’s jump to Jupyter!

Some Resources

- The Qiskit textbook
- List of qiskit resources
- Quantum computing languages landscape
- Qiskit tutorial

References:
https://qiskit.org/textbook/preface.html
TCS QubitxQubit