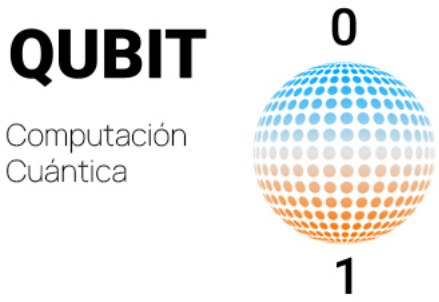


Computación Cuántica

Curso Qureca

Año: 2025/26

M1: Historical context

The power of Grover's algorithm lies in its ability to amplify the probability of finding the correct solution through quantum interference. It uses a quantum oracle to mark the desired item and then applies a series of quantum operations to increase the amplitude of the marked state. This process, known as amplitude amplification, allows the algorithm to find the target item with high probability after only $\sqrt{N}$ iterations. For large databases, this quadratic speedup can translate to significant time savings, making previously intractable search problems feasible.

The four branches of quantum technologies

The four branches of quantum technologies: quantum computing, quantum communication, quantum simulation, and quantum sensing.

- ***Quantum computing*** represents a paradigm shift in computational technology, moving beyond classical binary systems to harness quantum mechanical phenomena. At its core are qubits – quantum bits that differ fundamentally from classical bits. While classical bits store information as either 0 or 1, qubits exploit quantum superposition, allowing them to exist in a combination of both states simultaneously. This property, combined with quantum entanglement (where qubits remain correlated across distances), enables quantum computers to process vast amounts of information in parallel.
  This means that tasks that would take current computers years or even centuries to complete could be accomplished in a fraction of the time with quantum computing. Qubits can be physically realised through various quantum systems:

  - Photonic qubits: Encode information in photon properties like polarization or path.
  - Superconducting circuits: Use microwave-frequency currents in cooled chips.
  - Trapped ions: Leverage atomic energy states.
  - Quantum dots: Manipulate electron spin states.

- ***Quantum communication*** utilises the principles of quantum mechanics to secure data transmission. One of its main applications is quantum key distribution (QKD), which ensures that encryption keys are shared securely by encoding information into quantum states, ensuring provably-secure key sharing.

  Another important tool to be familiar with are quantum repeaters. They provide the same functionality as classical repeaters (ensuring that information traveling long distances are not weakened or distorted) but within a quantum network. Quantum information cannot be copied, so new techniques in order to amplify signals are currently being researched.

  Quantum communication can be very important to key industries around the world, but healthcare is probably one of the most important. With the vast amounts of sensitive patient data and confidential medical records stored and transmitted electronically, ensuring the privacy and integrity of this information is of paramount importance. This heightened security is particularly crucial in healthcare, where protecting patient privacy and preventing unauthorised access to medical data are imperative. Quantum communication not only fortifies the confidentiality of patient

information but also guards against emerging threats in the era of advanced cyberattacks. These advancements enable secure quantum information transmission over long distances, vital for industries requiring high confidentiality, such as finance, defense, and healthcare.

- ***Quantum simulation*** is a pioneering technique for modelling and understanding the intricate interactions of quantum systems, providing valuable insights into their behaviour. This approach enables exploration of complex molecules, aiding in the development of new drugs and materials. Quantum simulation is also crucial for understanding the process of protein folding and other fundamental biological processes. Moreover, it has the potential to design high-temperature superconductors, revolutionising energy transmission by enabling electricity transmission with zero resistance.

  One use case for quantum simulation is in simulating complex systems, for example, chemistry simulations. Most complex systems calculations are run on large computer clusters but take depending on the size of the simulation, can take weeks of calculations. By utilising quantum computers that time and accuracy can be dramatically improved. They can be also be used to optimise logistics and operations in hospitals, accelerate genomics, and improve radiation treatment plans.

- ***Quantum sensing*** marks a significant advancement in sensing capabilities, offering unmatched levels of sensitivity, selectivity, and efficiency. By leveraging quantum phenomena, quantum sensors facilitate advanced metrology applications. They excel in precision measurements of parameters like temperature, number of particles, gravity, time, and pressure.

  For example, their exceptional sensitivity to magnetic fields has made them invaluable for detecting magnetic biomarkers – molecules or materials that indicate specific biological conditions or processes. This quantum-enhanced sensing capability allows for the detection of extremely weak magnetic signals emitted by cells and tissues, enabling earlier and more accurate disease diagnosis. For instance, quantum sensors based on nitrogen-vacancy centres in diamond can detect minute magnetic fields produced by individual neurons or cancer cells, offering unprecedented insights into brain function and tumor development. Another important quantum sensor is the quantum radar. The quantum radar is an enhanced version of the current radar. By using entangled photons and comparing the returned photon and a stored photon, a quantum sensor can create a precise scan of an area. This approach would make radar virtually untraceable and become resilient to jamming and spoofing.

Algorithms

Peter Shor introduced ***Shor's algorithm***, capable of breaking modern encryption systems, and developed quantum error correction codes, crucial for maintaining quantum state integrity.

Grover's algorithm revolutionised database searches, and Seth Lloyd expanded quantum computing applications with algorithms for simulating quantum systems.

Types of qubits

Un **qubit** (abreviatura de *quantum bit*) es la unidad básica de información cuántica, análoga al **bit** en la computación clásica, pero con propiedades mucho más potentes gracias a la **mecánica cuántica**. En la computación clásica, un **bit** puede estar en uno de dos estados:

$$0 \quad \text{o} \quad 1$$

Por ejemplo, una bombilla puede estar apagada (0) o encendida (1). Toda la información digital se construye combinando muchos bits.

Un **qubit** puede estar en una **superposición** de los dos estados 0 y 1 al mismo tiempo. Matemáticamente se expresa como:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \cos\theta\,|0\rangle + \sin\theta\,|1\rangle$$

donde:

- $|0\rangle$ y $|1\rangle$ son los estados base,
- α y β son números complejos llamados *amplitudes*,
- y se cumple que $|\alpha|^2 + |\beta|^2 = 1$.

Esto significa que hasta que se mide el qubit, no está solo en 0 o 1, sino en una **combinación** de ambos. Al medirlo, el qubit colapsa a:

$$\begin{cases} 0 & \text{con probabilidad } |\alpha|^2, \\ 1 & \text{con probabilidad } |\beta|^2. \end{cases}$$

Este principio permite que un sistema de n qubits represente 2^n estados a la vez, otorgando un gran paralelismo computacional.

Dos o más qubits pueden estar **entrelazados**, lo que significa que el estado de uno depende instantáneamente del estado del otro, sin importar la distancia. Este fenómeno es clave para la ventaja cuántica en cálculos, comunicación y criptografía.

Un qubit puede representarse como un punto en la **esfera de Bloch**, donde:

- El polo norte representa el estado $|0\rangle$,
- El polo sur representa el estado $|1\rangle$,
- Cualquier punto sobre la superficie representa una superposición distinta.

Esto muestra que el qubit no solo tiene dos valores (0 y 1), sino una infinita gama de estados intermedios.

- ***Superconducting qubits*** are tuned by a SQUID (superconducting quantum interference device), based on changing the energy levels of a Josephson junction. These electronic components are when two superconducting materials are separated by an insulator, this causes electron energy levels to separate at specific intervals, great for being used as qubits.

  These qubits are also easily reproduced because they are based on known fabrication techniques of semiconductors. As well as easily connected to their nearest neighbors through known RF engineering techniques.

  The biggest drawback of these qubits is the temperature requirement. Superconducting states still require below 1 Kelvin (or -272.15° C) temperatures to maintain themselves.

- ***Trapped ion qubits*** are another promising avenue. These qubits are created using individually confined and manipulated charged particles, typically single ions of elements such as calcium, ytterbium, or beryllium. The ions are held in place by electromagnetic fields and controlled using precisely targeted lasers, allowing for highly accurate quantum operations.

  Since the atoms are in an isolated environment, the coherence times are much long than compared to superconducting qubits. The downside of using these singular atoms is that they can be easily disturbed if they are not in that perfectly isolated environment. Furthermore, the enclosure that they sit in requires complex control systems to operate.

- ***Photonic qubits*** encode information onto individual photons with a pre-set chip. By encoding information onto the photon's polarisation the photon can act as a qubit. Photons are efficient and fast but require optics to accomplish this hence a preset chip with the optics embedded in.

  Photonic chips also have low decoherence through their minimal interaction with the environment, which also means they have long coherence times. The two downsides to these preset chips are that it is hard to achieve strong interactions between photons to achieve greater complexity within calculations, and that detecting single photons can be resource demanding.

- ***Neutral atom qubits*** have no charge, making them an interesting alternative to trapped ion qubits. These work by using tiny changes in the energy levels of electrons when they're exposed to a magnetic field. Elements, such as rubidium and caesium, are placed into an optical array where the energy levels of electrons within the element are manipulated by light to store information. Since they don't have charge, they can be packed together tightly, meaning they can be scaled up easily. And the precise control over the hyperfine states means that quantum gate operations have low error rates.

  Current challenges with these qubits are developing strong and controllable interactions between neutral atoms for entanglement and gate operations.

- The final hardware platform we will cover is ***silicon spin qubits***. These qubits utilise the spin state of individual electrons in a silicon crystal as the qubits. Silicon is a widely used and well-understood material in the electronics industry. This familiarity offers significant advantages for quantum computing. The extensive knowledge and existing infrastructure for silicon manufacturing make it easier to create and work with silicon-based qubits. Additionally, silicon's properties allow for the potential integration of many qubits on a single chip, which is crucial for scaling up quantum computers.

  Silicon crystals have a low noise environment on the inside, meaning long coherence times and robust spin states can be achieved. However, due to this robustness, coupling between two qubits remains a challenge, and so, two-qubit gate operations are unreliable.

M2: Mathematical Formalism and Fundamental Concepts

We can define 4 operators for a single bit: Identity $I(0) = 0, I(1) = 1$, el $NOT(0) = 1, NOT(1) = 0$, y los constantes $ZERO(0,1) = 0, ONE(0,1) = 0$. After applying Identity or NOT operator, we can easily determine the initial value by checking the final value (reversible). Not for the others (irreversible).

If we combine both bits as a single system, then what is the state of the combined system?

In total, we have four different states. We can name them as follows:

00: both bits are in states 0 01: the first bit is in state 0 and the second bit is in state 1 10: the first bit is in state 1 and the second bit is in state 0 11: both bits are in states 1

Bits are know your bases. $\varphi_1 = (|0\rangle_1 + |1\rangle_1)$ and $\varphi_2 = (|0\rangle_2 + |1\rangle_2)$. If we want to apply and operator just to one of it: $M_2\varphi == (Id_1\varphi_1 \times M_2\varphi_2))$

Quantum coin flip

n the first experiment of quantum coin-flipping, we aim to trace the behavior of a photon, which acts as our coin. The setup involves sending photons to a beam splitter, which serves as the mechanism to flip the photon. When a photon encounters the beam splitter, it can either be transmitted or reflected, akin to the two possible outcomes of a coin flip. To measure these outcomes, we use two photon detectors positioned to capture the transmitted and reflected photons. These detectors act as our eyes allowing us to observe and record the behavior of the photon. By analyzing the data from the detectors, we can understand the probabilistic nature of the photon's path.

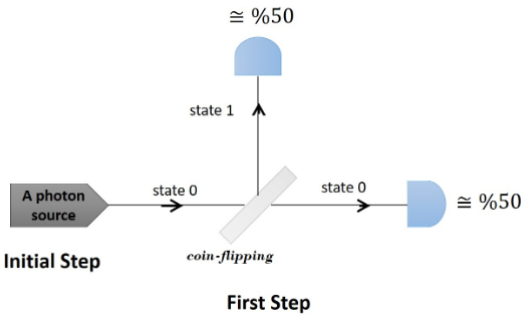
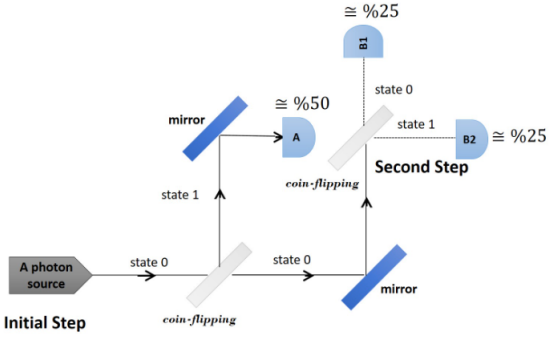


Figura 1: If the splitter is fair, we have 1/2 probabilities of each state.



Hadamart Operator

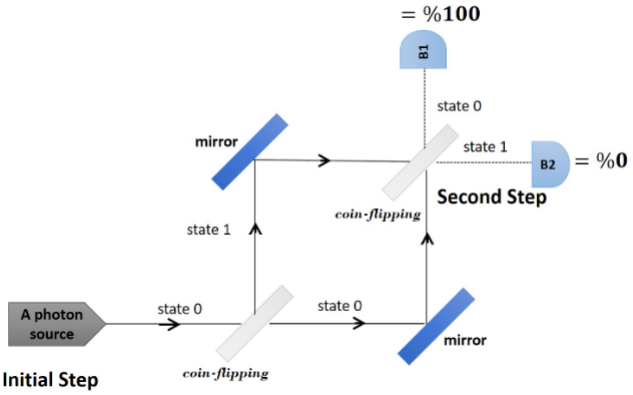


Figura 2: In classical physics It will be half probability each, but not in quantum mechanics once you skip the middle measurement.

El ***Hadamard operator***, generalmente denotado como H es una puerta cuántica que actúa sobre un solo qubit. Su función principal es convertir un qubit de un estado $|0\rangle$ o $|1\rangle$ en una superposición equitativa de ambos estados. En mi caso es el beam splitter.

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

It gives you an orthonormal basis with $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, with $\langle +||-\rangle = 0$.

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram

q = QuantumRegister(1, "qreg") (\textit{Esto es un qubit})
c = ClassicalRegister(1, "creg") (\textit{Esto un bit normal que recogerá 1 o 0
\\
una vez midamos el qubit y colapse})

qc = QuantumCircuit(q, c) (\textit{Simplemente un circuito})
qc.h(q[0]) (\textit{Aplicamos Havard (superposición de ambos estados al qubit)})
qc.measure(q, c) (\textit{Guarda en c})

qc.draw(output='mpl')

job = execute(qc, Aer.get_backend('qasm_simulator'), shots=10000)
counts = job.result().get_counts(qc)
print(counts)

n_zeros = counts['0']
n_ones = counts['1']
print("State 0 is observed with frequency %", 100*n_zeros/(n_zeros+n_ones))
print("State 1 is observed with frequency %", 100*n_ones/(n_zeros+n_ones))

plot_histogram(counts)
```

A quantum system can be in more than one state with nonzero amplitudes. Then, we say that our system is in a superposition of these states. When evolving from a superposition, the resulting transitions may affect each other constructively and destructively. This happens because of having opposite sign transition amplitudes. Otherwise, all nonzero transitions are added up to each other as in probabilistic systems. After the measurement, the system collapses to the observed state, and so the system is no longer in a superposition. Thus, the information kept in a superposition is lost.

Esto es lo que pasa entre el experimento 2 y 3, que pierdes la superposición. En el 3 aplicas Hadamard dos veces y mides, en el 2 mides entre medias de aplicarlas.

Operations on the Unit Circle

We are starting with $|0\rangle$ and $|+\rangle$: Te dibuja el circulito (hay bastantes ejemplos).

```
quantum_file = path_files+"quantum.py"
%run $quantum_file

draw_qubit()

sqrttwo=2**0.5

draw_quantum_state(1,0,"")

draw_quantum_state(1/sqrttwo,1/sqrttwo,"|+>")

# drawing the angle with |0>-axis
from matplotlib.pyplot import gca, text
from matplotlib.patches import Arc
gca().add_patch( Arc((0,0),0.4,0.4,angle=0,theta1=0,theta2=45) )
text(0.08,0.05,'.',fontsize=30)
text(0.21,0.09,'\u03C0/4')
show_plt()
```

Any real-valued quantum state is a point in the unit circle, and it can be described by an angle $|v\rangle = (\cos\theta, \sin\theta)$.

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram
from math import pi

# we define a quantum circuit with one qubit and one bit
q =  QuantumRegister(1) # quantum register with a single qubit
c = ClassicalRegister(1) # classical register with a single bit
qc = QuantumCircuit(q,c) # quantum circuit with quantum and classical registers

# angle of rotation in radian
rotation_angle = 2*pi/3

# rotate the qubit with rotation_angle
qc.ry(2*rotation_angle,q[0])

# measure the qubit
qc.measure(q,c)
```

```
    # draw the circuit
    qc.draw(output='mpl')

    # execute the program 1000 times
    job = execute(qc,Aer.get_backend('qasm_simulator'),shots=1000)

    # print the results
    counts = job.result().get_counts(qc)
    print(counts)

    # draw the histogram
    plot_histogram(counts)
```

Te salen como 800 cuentas en el estado $|1\rangle$ y 200 en $|0\rangle$. Lo único que hacen estos códigos es generar un qubit que dependan las probabilidades de θ.

Two qubits

Our states are know: $|00\rangle, |10\rangle, |01\rangle, |11\rangle \rightarrow (1,0,0,0), (0,1,0,0), (0,0,0,1), (0,0,0,1)$.

```
    from qiskit import QuantumCircuit

# remark the concise representation of a quantum circuit
qc = QuantumCircuit(2)

qc.h(0)
qc.h(1)

qc.draw(output='mpl',reverse_bits=True)
job = execute(circuit, Aer.get_backend('unitary_simulator'),optimization_level=0)
current_unitary = job.result().get_unitary(circuit, decimals=3).data
print(current_unitary)
from qiskit import execute, Aer

job = execute(qc, Aer.get_backend('unitary_simulator'),shots=1,optimization_level=0)
current_unitary = job.result().get_unitary(qc, decimals=3).data
for row in current_unitary:
    column = ""
    for entry in row:
        column = column + str(entry.real) + " "
    print(column)

    0.5 0.5 0.5 0.5
    0.5 -0.5 0.5 -0.5
    0.5 0.5 -0.5 -0.5
    0.5 -0.5 -0.5 0.5
```

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

Case 1: Let's find $H^{\otimes 2}|00\rangle$ (in three different ways)

- Direct matrix-vector multiplication:

$$H^{\otimes 2}|00\rangle = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$$

Figura 3: Literalmente nos está dando $H^2 = H \times H$ en la base nueva.

Entanglement and Superdense Coding

Peter and Charles has two initial qubits, both in $|0\rangle$. Then Peter applies Hadamart on hims, so now his state is: $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. The combine state of both is: $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$ gave by tensorial product.

If we know apply CNOT: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ Our new quantum state is: $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.
Ahora el qubit de Charles va a colapsar a la vez que lo haga el de Peter al medir. Experimental results have confirmed that this happens even if there is a physical distance between Asja's and Balvis' qubits. It seems correlated quantum particles can affect each other instantly, even if they are in the different part of the universe. If two qubits are correlated in this way, then we say that they are ***entangled***. After having the entanglement, Balvis takes his qubit and goes away.

Asja will send two classical bits of information by only sending her qubit. If the state is

$$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle,$$

then it is entangled if

$$ad - bc \neq 0.$$

$H =$ Hadamard gate $\rightarrow$ crea superposición.

$X =$ Pauli-X gate $\rightarrow$ actúa como un "NOT" cuántico ($|0\rangle \leftrightarrow |1\rangle$).

$Z =$ Pauli-Z gate $\rightarrow$ cambia la fase del estado $|1\rangle$.

Quantum Teleportation

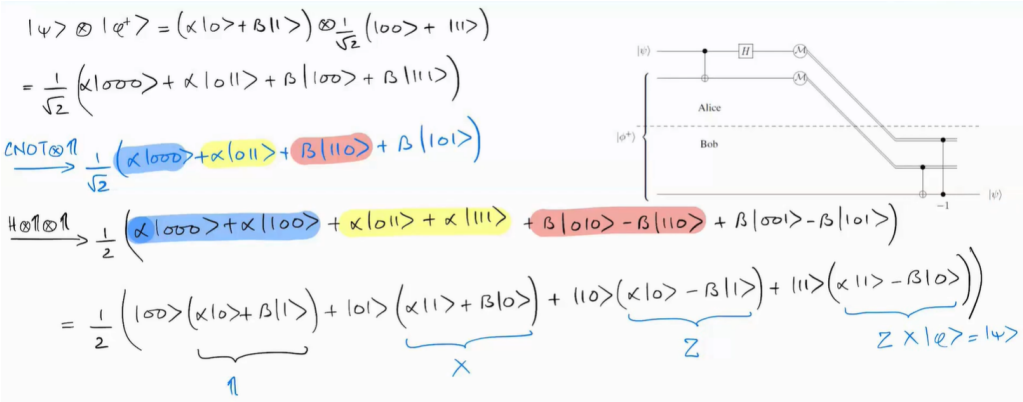


Figura 4: Asja tiene dos qubits, uno en un estado $\alpha|0\rangle + \beta|1\rangle$ y otro en entrelazamiento con otro de Pedro. Al componer la función de onda total tendremos cuatro estados posibles, dado que los qubits entrelazados colpasan juntos y dan el mismo valor al haber aplicado el beam splitter. Si Asja aplica CNOT y seguido Hadamart en su qubit propio y hace una medición, tendrá un 1/4 de probabilidades de dar con el estado $|00\rangle, |10\rangle, |11\rangle, |01\rangle$. Una vez mida, el qubit de Pedro viene descrito por la función correspondiente. Es decir, ha pasado las probabilidades de su qubit original al de Pedro. So, using only two bits of classical communication and prior shared entanglement Asja is able to transfer an unknown quantum state to Balvis.

Test del tema 2

1. Question

Let $v = \begin{pmatrix} 2 \\ \sqrt{7} \\ 5 \end{pmatrix}$ be a vector. If we normalize the vector v then the new normalized vector is represented as follows.

Figura 5: $(\frac{1}{3}, \frac{\sqrt{7}}{6}, \frac{5}{6})$

By using the quantum teleportation protocol given our notebooks, Asja is teleporting the state $\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \in \mathbb{R}^2$ to Balvis.

Immediately after Asja's measurement, Balvis qubit will be in a mixture of pure states (before post-processing). If this measurement result is '01' or '10', what is this mixture?

Figura 6: Ver el ejercicio de arriba

2. Question

We have a three state quantum system. If the system is in the quantum state

$|u> = \begin{pmatrix} x \\ \frac{1}{\sqrt{7}} \\ \frac{2}{\sqrt{7}} \end{pmatrix} \in \mathbb{R}^3$

what is the probability of being in the first state?

Figura 7: 2/7

3. Question

Given a probabilistic system of two bits and a operator,

$$\begin{pmatrix} 0.10 & 0.20 & 0.30 & 0.40 \\ 0.35 & 0.25 & 0.15 & 0.05 \\ 0.05 & 0.15 & 0.25 & 0.35 \\ 0.50 & 0.40 & 0.30 & 0.20 \end{pmatrix},$$

what is the probability of going from state $\lceil 01 \rfloor$ to state $\lceil 10 \rfloor$?

Figura 8: 0.15

We have 1000 copies of the identical qubit in state $\begin{pmatrix} cos\theta \\ sin\theta \end{pmatrix}$, where $\theta \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$.

After measuring 1000 copies, we observe |0> 201 times and state |1> 799 times.

Which one of the followings can be more likely a value of θ in degree?

Figura 9: -63º

5. Question

We have a composite system with two probabilistic bits. Let

$v = \frac{2}{5}\lceil 00 \rfloor + \frac{1}{5}\lceil 10 \rfloor + \frac{2}{5}\lceil 11 \rfloor$ be its probabilistic state. Which one of the following represents this state?

Figura 10: $(2/5, 0, 1/5, 2/5)$

If the following matrix U is unitary, then what is the value of a?

$$U = \begin{pmatrix} -\frac{2}{\sqrt{5}} & \frac{i}{\sqrt{5}} \\ a & \frac{2}{\sqrt{5}} \end{pmatrix}$$

$$\frac{i}{\sqrt{5}}$$

Figura 11: $UU^\dagger = 1$

We have a qubit in state |0>.

The rotations $R\left(\frac{\pi}{3}\right)$ and $R\left(-\frac{\pi}{6}\right)$ are applied m and n times, respectively.

If the final state is -|1> , what can be the values of (m, n)?

$$(20, 11)$$

Which one of the following pairs of quantum states is perfectly distinguishable?

$$\left(\sqrt{\frac{5}{7}}|0> -\sqrt{\frac{2}{7}}|1>, -\sqrt{\frac{2}{7}}|0> -\sqrt{\frac{5}{7}}|1>\right)$$

$$\left(\sqrt{\frac{5}{7}}|0> +\sqrt{\frac{2}{7}}|1>, -\sqrt{\frac{2}{7}}|0> -\sqrt{\frac{5}{7}}|1>\right)$$

Figura 12: Cosas de signos en los productos

What is the matrix form of the reflection having the line of reflection $y = -x$?

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

10. Question

Which one is the most likely outcome of the following code?

```
from random import randrange
heads = tails = 0
for i in range(100):
    if randrange(4)==randrange(4):
        heads = heads + 1
    else:
        tails = tails + 1
print(heads)
```

4

27

Figura 13: Genera 100 números aleatorios, y si salen (1,2,3,4) va sumando o no se que (es un 25 por ciento)

If we execute the following program 1000 times, what is the most likely count of observing '0'?

625

125

Figura 14: Dos H se anulan. Verdaderamente es un 50 por ciento (500) pero se acerca más 625.

We have two qubits as $q_1 \otimes q_0$ in state

$$\sqrt{\frac{1}{10}}\,|00> - \sqrt{\frac{2}{10}}\,|01> - \sqrt{\frac{3}{10}}\,|10> - \sqrt{\frac{4}{10}}\,|11>$$

If we measure only q_1, which one of the following mixtures is obtained?

Figura 15: $(pr = 3/10, \sqrt{1/3}\,|00\rangle - \sqrt{2/3}\,|01\rangle)(pr = 7/10, -\sqrt{3/10}\,|10\rangle - \sqrt{4/10}\,|11\rangle)$

13. Question

In which one of the following states, the qubits are not entangled?
Hint: One may find it easier to work with the state vector.

$$\frac{1}{2}(|00> -|01> -|10> +|11>)$$

Figura 16: Ni zorra

14. Question

Let |u> be a quantum state on the unit circle with angle θ.

We apply $Ref(\theta_1)$ and then $Ref(\theta_2)$.

What is the angle of the final state?

Figura 17: $-2\theta_1 + 2\theta_2 + \theta$

15. Question

We have four qubits, say q_0, q_1, q_2, q_3, initially all in zero states.

We apply the X operators to both qubits q_0 and q_2.

For the rest of qubits, we apply either identity operator or X operator.

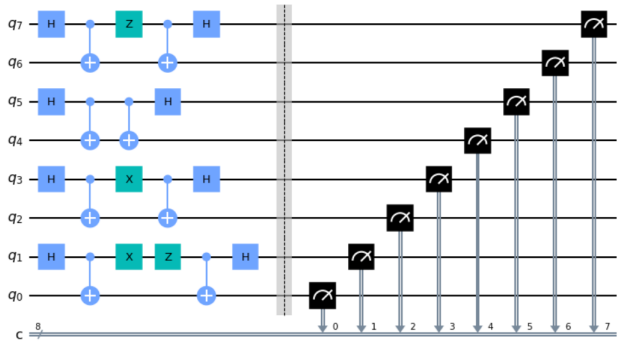
After making a measurement, we read the values from the qubits q_0, q_1, q_2, q_3 as

b_0, b_1, b_2, b_3, respectively.

If $b = b_3 b_2 b_1 b_0$ is a binary number, which one of the following decimal numbers is

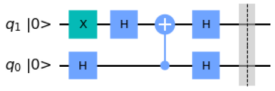
not possible for the value of b?

Figura 18: Poisbles: 1010(5),1110(13),1111(15),1011(7) $\rightarrow$ 11 no está



18. Question

What is the state at the barrier?



Estado inicial: $|0\rangle_{q_1}|0\rangle_{q_0}$

X en $q_1 \Rightarrow |1\rangle_{q_1}|0\rangle_{q_0}$

H en ambos: $|-\rangle_{q_1}|+\rangle_{q_0}$

Después del CNOT (control q_0, target q_1) :

$|\psi'\rangle = |-\rangle_{q_1}|-\rangle_{q_0}$

H en ambos: $H|-\rangle = |1\rangle$

$\Rightarrow |\psi_{\text{final}}\rangle = |1\rangle_{q_1}|1\rangle_{q_0} = |11\rangle$

$$\boxed{|\psi_{\text{barrera}}\rangle = |11\rangle}$$

Definición CNOT: Si el control es $|1\rangle$, aplica X al target; si es $|0\rangle$, no hace nada.

$$|00\rangle \to |00\rangle, \qquad |01\rangle \to |01\rangle,$$
$$|10\rangle \to |11\rangle, \qquad |11\rangle \to |10\rangle$$

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

19. Question

We have a qubit in state |1> , We apply the operators X, H, X, H, X in order, where H and X are the Hadamard and NOT operators, respectively. What is the final state?

- |+>
- |0>
- |1>

$|\psi_0\rangle = |1\rangle$

$X|1\rangle = |0\rangle \quad \Rightarrow \quad |\psi_1\rangle = |0\rangle$

$H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}} = |+\rangle \quad \Rightarrow \quad |\psi_2\rangle = |+\rangle$

$X|+\rangle = |+\rangle \quad \Rightarrow \quad |\psi_3\rangle = |+\rangle$

$H|+\rangle = |0\rangle \quad \Rightarrow \quad |\psi_4\rangle = |0\rangle$

$X|0\rangle = |1\rangle \quad \Rightarrow \quad |\psi_5\rangle = |1\rangle$

$$\boxed{|\psi_{\text{final}}\rangle = |1\rangle}$$

M3: Fundamental Quantum Algorithms

This classification is important, as the quantum evolution operators are reversible (as long as the system is closed).

The Identity and NOT operators are two fundamental quantum operators.

Deutsch Algorithm

We want to know if the aplication $f : \{0,1\} \to \{0,1\}$ is balanced $f(0) \neq f(1)$ or constant $f(0) = f(1)$. Classically, we should make two measure to analyse it. Lets remember that every function can be express as an ***Unitary Operator*** U_f.

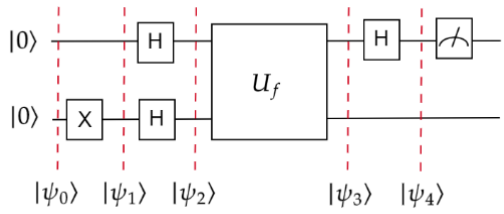


Figura 19: Measure the first qubit. If it is 0 then f is constant. If it is 1, then f is balanced.

Comenzamos con el estado inicial:

$$|\psi_0\rangle = |0\rangle\,|0\rangle\,.$$

Aplicamos una compuerta X al segundo qubit:

$$|\psi_1\rangle = |0\rangle\,|1\rangle\,.$$

Luego aplicamos una compuerta de Hadamard H a ambos qubits, obteniendo:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|-\rangle = \frac{1}{\sqrt{2}}\,|0\rangle\,|-\rangle + \frac{1}{\sqrt{2}}\,|1\rangle\,|-\rangle\,.$$

A continuación aplicamos el operador U_f:

$$|\psi_3\rangle = U_f\,|\psi_2\rangle = \frac{1}{\sqrt{2}}(-1)^{f(0)}\,|0\rangle\,|-\rangle + \frac{1}{\sqrt{2}}(-1)^{f(1)}\,|1\rangle\,|-\rangle\,.$$

Por el efecto de *phase kickback*, el segundo qubit queda inalterado y toda la información de f se transfiere al primero:

$$|\psi_3\rangle = \left(\frac{1}{\sqrt{2}}\left[(-1)^{f(0)}\,|0\rangle + (-1)^{f(1)}\,|1\rangle\right]\right)|-\rangle\,.$$

Aplicamos una compuerta H al primer qubit:

$$H\,|\psi_3\rangle = \frac{1}{2}\left[\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle + \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle\right]|-\rangle\,.$$

Finalmente, analizamos dos casos:

- **Caso constante:** si $f(0) = f(1)$, entonces

$$|\psi_4\rangle = (-1)^{f(0)}\,|0\rangle\,|-\rangle\,.$$

  La medición del primer qubit da $|0\rangle$ con probabilidad 1.

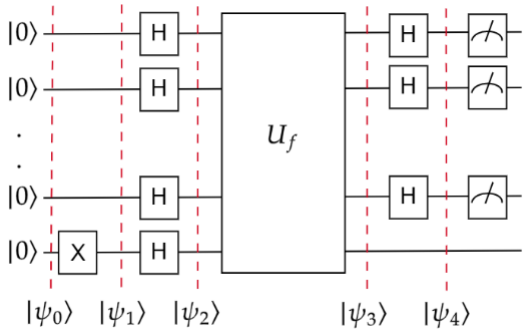
- **Caso balanceado:** si $f(0) \neq f(1)$, entonces

$$|\psi_4\rangle = (-1)^{f(0)} |1\rangle |-\rangle .$$

  La medición del primer qubit da $|1\rangle$ con probabilidad 1.

Por tanto, con una única evaluación de f, podemos determinar con certeza si f es constante o balanceada. Este es el principio del **algoritmo de Deutsch**, el primer ejemplo de una ventaja cuántica sobre los métodos clásicos (que requieren dos consultas).

Deutsch-Jozsa Algorithm



Consideremos una función

$$f : \{0,1\}^n \to \{0,1\},$$

que toma una cadena binaria de longitud n como entrada y produce 0 o 1 como salida. La función f es:

- **Constante**, si todos los valores de entrada se mapean al mismo valor (0 o 1).
- **Balanceada**, si la mitad de las entradas se mapean a 0 y la otra mitad a 1.

Definición del operador cuántico

El oráculo cuántico U_f se define como:

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle .$$

Cuando el segundo qubit se encuentra en el estado $|-\rangle$, se produce el *efecto de retroceso de fase* (*phase kickback*):

$$U_f |x\rangle |-\rangle = (-1)^{f(x)} |x\rangle |-\rangle \tag{1}$$

Notación para registros múltiples

Un registro de n qubits en el estado $|0\rangle$ se denota como:

$$|0\rangle^{\otimes n} = |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle .$$

De manera similar, el operador de Hadamard sobre n qubits se expresa como $H^{\otimes n}$.

Transformación de Hadamard generalizada

Sabemos que:

$$H|x_i\rangle = \frac{1}{\sqrt{2}} \sum_{z_i \in \{0,1\}} (-1)^{x_i z_i} |z_i\rangle \tag{2}$$

Por lo tanto, para un registro de n qubits:

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle \tag{3}$$

donde $x \cdot z = \sum_{i=1}^{n} x_i z_i \pmod 2$.

Descripción del algoritmo

1. Iniciamos con el estado:
$$|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle .$$

2. Aplicamos una compuerta X al último qubit:
$$|\psi_1\rangle = |0\rangle^{\otimes n} |1\rangle .$$

3. ***Aplicamos una compuerta Hadamard H al último qubit para obtener $|-\rangle$, y $H^{\otimes n}$ a los n primeros qubits:***
$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |-\rangle .$$

4. Aplicamos el oráculo U_f:
$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle |-\rangle .$$

5. Aplicamos nuevamente $H^{\otimes n}$ a los n primeros qubits:
$$|\psi_4\rangle = \frac{1}{2^n} \sum_{z=0}^{2^n-1} \left[\sum_{x=0}^{2^n-1} (-1)^{x \cdot z + f(x)} \right] |z\rangle |-\rangle .$$

6. Medimos los primeros n qubits.

Análisis de resultados

La amplitud correspondiente al estado $|z = 0^n\rangle$ es:

$$\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)}.$$

- Si f es **constante**, todos los términos tienen el mismo signo y la suma da ± 1, por lo que se observa $|0\rangle^{\otimes n}$ con probabilidad 1.
- Si f es **balanceada**, la mitad de los términos son positivos y la otra mitad negativos, cancelándose entre sí. En este caso, la probabilidad de observar $|0\rangle^{\otimes n}$ es 0.

Conclusión

El algoritmo de Deutsch–Jozsa permite determinar si una función f es constante o balanceada mediante una única llamada al oráculo U_f. En comparación, un algoritmo clásico requeriría un número de consultas que crece exponencialmente con n. Este resultado representa la primera demostración de una **ventaja cuántica exponencial** dentro del modelo de cómputo con oráculo.

Fijarse que siempre se tiene un bit en 1 para aplicar H y tener $|-\rangle$, de tal forma que al aplicar H nunca entre en juego por phase kickback.

Bernstein-Vazirani Algorithm

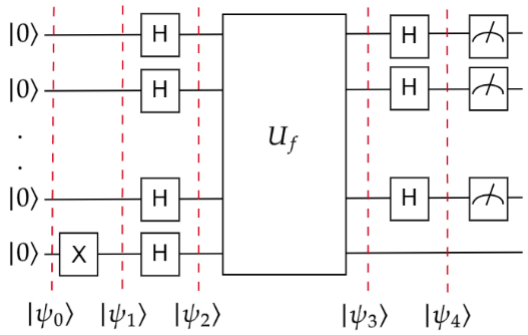


Figura 20: Caption

Sea una función

$$f : \{0,1\}^n \to \{0,1\},$$

definida como

$$f(x) = x \cdot s,$$

donde $s \in \{0,1\}^n$ es una cadena binaria *secreta* y $x \cdot s$ representa el producto interno módulo 2:

$$x \cdot s = \sum_{i=1}^{n} x_i s_i \pmod 2.$$

El objetivo es determinar la cadena s.

Ejemplo

Por ejemplo, si $x = 1000$ y $s = 1010$, entonces:

$$x \cdot s = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 1 \pmod 2 = 1.$$

Supongamos $n = 2$ y:

$$f(00) = 0,$$
$$f(01) = 1,$$
$$f(10) = 0,$$
$$f(11) = 1,$$

entonces $s = 01$.

Operador cuántico

El oráculo cuántico U_f implementa la función f de la siguiente forma:

$$U_f : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus (x \cdot s)\rangle .$$

Cuando el segundo qubit está en el estado $|-\rangle$, el *efecto de retroceso de fase* implica:

$$U_f |x\rangle |-\rangle = (-1)^{x \cdot s} |x\rangle |-\rangle .$$

Algoritmo

El circuito cuántico es esencialmente el mismo que en el algoritmo de Deutsch–Jozsa, con $n + 1$ qubits.

1. Inicializamos el sistema en:
$$|\psi_0\rangle = |0\rangle^{\otimes n} |0\rangle .$$
2. Aplicamos una compuerta X al último qubit:
$$|\psi_1\rangle = |0\rangle^{\otimes n} |1\rangle .$$
3. Aplicamos Hadamard H al último qubit para crear $|-\rangle$, y $H^{\otimes n}$ a los n primeros:
$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |-\rangle .$$
4. Aplicamos el oráculo U_f:
$$|\psi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{x \cdot s} |x\rangle |-\rangle .$$
5. Aplicamos nuevamente $H^{\otimes n}$ a los primeros n qubits.

Análisis

Recordando que el operador de Hadamard cumple:

$$H^{\otimes n} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle ,$$

y que $H^{\otimes n}$ es su propio inverso, notamos que:

$$|\psi_3\rangle = H^{\otimes n} |s\rangle .$$

Por tanto, al aplicar nuevamente $H^{\otimes n}$, obtenemos:

$$|\psi_4\rangle = |s\rangle .$$

Resultado

Al medir los primeros n qubits, se obtiene la cadena secreta s con probabilidad 1. Este algoritmo es determinista y exacto: requiere sólo **una consulta** al oráculo U_f, mientras que el algoritmo clásico necesitaría n consultas.

Conclusión

El algoritmo de Bernstein–Vazirani demuestra una clara **ventaja cuántica lineal** sobre los métodos clásicos, al descubrir la cadena s con una sola evaluación de f. Aunque la mejora no es exponencial, este algoritmo ilustra cómo el paralelismo cuántico puede codificar información global sobre una función en una única medición.

Simon Algorithm

Given a function (implemented by an oracle)

$$f : \{0,1\}^n \to \{0,1\}^n,$$

such that

$$f(x) = f(x \oplus s),$$

for some secret string $s \in \{0,1\}^n$, the goal is to determine the value of s by making as few queries to $f(x)$ as possible.

The string s is referred to as our **secret string** (or hidden string) which we aim to decipher.

Bitwise XOR

The bitwise XOR operator $\oplus$ (also known as bitwise addition modulo 2) takes two bit strings as inputs and performs an exclusive OR operation on every bit pair from the two bit strings.

The XOR operator returns 1 if the bit pair is different and 0 if the bit pair is the same.

Applying the bitwise XOR operator to a random bit string $r \in \{0,1\}^n$ and the zero bit string $z = 0^n$ will return r as the output.

XOR Table for 2 bits

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

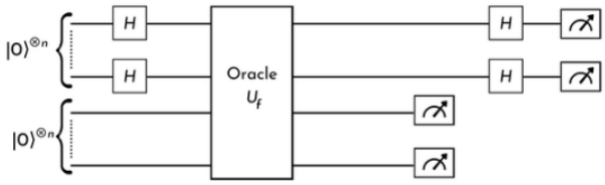
Examples

Consider the following example where bit strings are represented as vectors:

$$\begin{aligned} a_1 &= 101, \\ a_2 &= 011, \\ a_3 &= 100, \\ a_4 &= 000. \end{aligned}$$

Then:

$$\begin{aligned} a_1 \oplus a_2 &= (1,0,1) \oplus (0,1,1) = (1 \oplus 0, 0 \oplus 1, 1 \oplus 1) = 110, \\ a_1 \oplus a_3 &= (1,0,1) \oplus (1,0,0) = (1 \oplus 1, 0 \oplus 0, 1 \oplus 0) = 001, \\ a_2 \oplus a_3 &= (0,1,1) \oplus (1,0,0) = (0 \oplus 1, 1 \oplus 0, 1 \oplus 0) = 111, \\ a_1 \oplus a_4 &= (1,0,1) \oplus (0,0,0) = (1 \oplus 0, 0 \oplus 0, 1 \oplus 0) = 101. \end{aligned}$$



Back to the Problem

Now that we understand the bitwise XOR operation, let's return to our problem statement for a 3-bit string.

We investigate a function f which satisfies $f(x) = f(x \oplus s)$, where $s = 101$.

In Simon's problem, our goal is to find s making as few queries to f as possible. The secret string s can also be considered as a "mask" which hides the underlying encoding.

As long as s is not the zero bit string, the function is two-to-one, i.e., mapping two elements from the domain to one element in the range.

You may also observe that for two inputs x_1 and x_2 for which $f(x_1) = f(x_2)$, we have

$$x_1 \oplus x_2 = s.$$

Note: Simon's problem is often defined as the problem of determining whether f is two-to-one or one-to-one, in which case one needs to determine whether s is the zero string or not.

Classical Solution

Deterministic Approach

To find the value of s classically with absolute certainty, we can go through each input and compute $f(x)$ for different values of x to find two input strings that map to the same output string. Then, using the idea above, we can compute s.

If we're lucky, we solve the problem with our first two queries; if we're unlucky, we'll have to make $2^{n-1} + 1$ queries (just over half of all possible inputs) to solve the problem.

Thus, it generally takes exponential time to solve this problem classically.

Probabilistic Approach

Let us try to find the value of s making random queries to f.

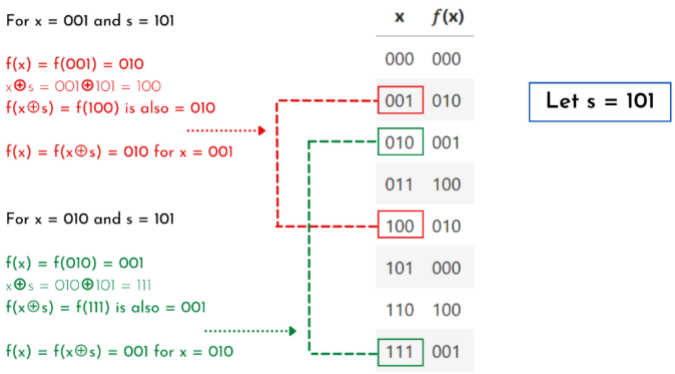
What is the probability of getting $f(x) = f(y)$ when you pick two random strings x and y? This is the well-known *birthday paradox* or *collision problem* in cryptography.

Encountering $f(x) = f(x \oplus s)$ when picking two random strings can be achieved if we randomly make approximately $2^{n/2}$ queries, given we do not encounter a one-to-one case.

This provides a good lower bound for the number of queries required. In general, it still takes exponential time to solve this problem in the probabilistic case.

As mentioned, there is also a deterministic algorithm for commutative/abelian groups that achieves the same query complexity as the best probabilistic algorithm. The algorithm for non-commutative/non-abelian groups comes within a polylogarithmic factor of the best probabilistic query complexity.

Quantum Solution



The claim is that the quantum analogue of Simon's algorithm solves this problem making

$$O(n)$$

queries, which is much better than the $O(2^{n/2})$ classical approaches.

The circuit is constructed and measured $O(n)$ times (until $n-1$ linearly independent equations are obtained) to get a list of n-bit strings $y \in \{0,1\}^n$. Each run of the circuit gives us a sample at random from the set satisfying the condition:

$$y \cdot s = 0 \quad (\text{mód } 2).$$

The rest of the algorithm is implemented classically.

After repeating the measurement n times, we have:

$$y_1, y_2, \ldots, y_n$$

and

$$s = (s_1, s_2, \ldots, s_n),$$

resulting in the following system of equations:

$$\begin{aligned} y_{1,1}s_1 + y_{1,2}s_2 + \cdots + y_{1,n}s_n &= 0, \\ y_{2,1}s_1 + y_{2,2}s_2 + \cdots + y_{2,n}s_n &= 0, \\ &\vdots \\ y_{n-1,1}s_1 + y_{n-1,2}s_2 + \cdots + y_{n-1,n}s_n &= 0. \end{aligned}$$

One of the solutions will be 0^n and another will be our secret message s, which is the value we are interested in.

We should keep in mind that $|0\rangle^{\otimes n}$ will always be a solution of this system, but what we are interested in is whether it is the only solution or not.

Any other solution will be our s. Given that we can solve this system of linear equations (using any method of our choice), we have arrived at our solution!

Grover Algorithm

Test del tema 3

2. Question

Let $b_1 = 011010$, $b_2 = 110010$ and $b_3 = 101010$, what are the results of the operations $b_1 \oplus b_2$ and $b_2 \oplus b_3$?

Figura 21: 101000 / 011000

3. Question

We have a quantum register with 3 qubits. We would like to create an oracle that marks the states 101 and 111. What should replace a below?

CCCZ = cirq.Z(qq[2]).controlled_by(qq[a])

- 0

Tenemos un registro de 3 qubits `qq` = `[qq[0], qq[1], qq[2]]`. Suponiendo la convención habitual de estado $|q_0 q_1 q_2\rangle$, los estados $|101\rangle$ y $|111\rangle$ comparten que $q_0 = 1$ y $q_2 = 1$, mientras que q_1 puede ser 0 o 1. Por tanto el oráculo debe aplicar una fase cuando $q_0 = 1$ y $q_2 = 1$, independientemente de q_1.

Una forma directa de hacerlo es una puerta CZ entre `qq[0]` (control) y `qq[2]` (target). En Cirq esto se expresa como

`cirq.Z(qq[2]).controlled_by(qq[0])`.

En la línea que diste, `CCCZ = cirq.Z(qq[2]).controlled_by(qq[a])`, debemos usar `a = 0`, es decir `qq[0]` como control.

Respuesta final: $\boxed{a = 0.}$

Código (Qiskit/Cirq-style):

```
# en Cirq (suponiendo qq es la lista de qubits):
CCCZ = cirq.Z(qq[2]).controlled_by(qq[0])
```

Nota: si tu convención de orden de qubits fuera diferente (p. ej. si `qq[0]` fuese el qubit menos significativo en la representación que usas), ajusta la elección del control en consecuencia; la idea clave es controlar por el qubit que corresponde a la primera posición del patrón 101.

4. Question

Let n=2. Suppose that the quantum state

$$\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\sum_{z\in\{0,1\}^n}(-1)^{x\cdot z+f(x)}|z\rangle$$

is measured, where $|z\rangle$ is a two-qubit state.. Which state is observed if f(x)=0 for all x?

$|00\rangle$

Se toma $n = 2$. El estado dado (interpretando la notación) es

$$\frac{1}{2^n}\sum_{x\in\{0,1\}^n}\sum_{z\in\{0,1\}^n}(-1)^{x\cdot z+f(x)}\,|z\rangle.$$

Para $n = 2$ y $f(x) = 0$ para todo x esto se convierte en

$$\frac{1}{4}\sum_{x\in\{0,1\}^2}\sum_{z\in\{0,1\}^2}(-1)^{x\cdot z}\,|z\rangle=\sum_{z\in\{0,1\}^2}\left(\frac{1}{4}\sum_{x\in\{0,1\}^2}(-1)^{x\cdot z}\right)|z\rangle.$$

Calculemos $S(z) := \sum_{x\in\{0,1\}^2}(-1)^{x\cdot z}$.

- Si $z = 00$ entonces $x \cdot z = 0$ para todo x, luego $S(00) = 4$.
- Si $z \neq 00$ (es decir $z \in \{01, 10, 11\}$) los términos $(-1)^{x\cdot z}$ se cancelan por pares y $S(z) = 0$.

Por tanto las amplitudes son

$$\frac{1}{4}S(z)=\begin{cases}1, & z=00,\\ 0, & z\in\{01,10,11\}.\end{cases}$$

Concluimos que el estado es exactamente $|00\rangle$, y por tanto la medición del registro de salida siempre da $|00\rangle$.

Estado observado: $|00\rangle$.

5. Question

When we apply the Hadamard gate to all the qubits of a state $|x\rangle$ we can expres

the result in the following way

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z=0}^{2^n-1} (-1)^{x \cdot z} |z\rangle$$

What are all the values that $|z\rangle$ can take if $|x\rangle$ is composed of three qubits, i.e.

$n = 3$?

Para $n = 3$ se tiene $z = 0, 1, \ldots, 2^3 - 1 = 7$. Los estados $|z\rangle$ son los ocho vectores base de 3 qubits:

$$|z\rangle \in \{|000\rangle,\ |001\rangle,\ |010\rangle,\ |011\rangle,\ |100\rangle,\ |101\rangle,\ |110\rangle,\ |111\rangle\}.$$

Así, para cualquier $x \in \{0,1\}^3$,

$$H^{\otimes 3}|x\rangle = \frac{1}{\sqrt{8}} \sum_{z=0}^{7} (-1)^{x \cdot z} \, |z\rangle,$$

donde $x \cdot z$ es el producto escalar bit a bit módulo 2.

6. Question

Given $s = 0110$, which oracle will return $f(x) = x \cdot s$?

```
def.oracle():
circuit= QuantumCircuit(5)
circuit.cx(1,4)
circuit.cx(2,4)
return circuit
```

7. Question

The code given below is the initialization step of Grover's Algorithm. There are 64 elements in the search list. What is (a,b)?

```
circuit = cirq.Circuit()
qubits = cirq.LineQubit.range([a])
circuit.append(cirq.[b].on_each(*qubits))
```

Figura 22: **Inicialización del algoritmo de Grover.** (1) Se utilizan $n = 6$ qubits, pues $2^n = 64$ elementos $\Rightarrow$ $\boxed{a = 6}$. (2) El estado inicial se prepara aplicando la puerta de Hadamard H a cada qubit, generando la superposición uniforme $\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$. En `Cirq`, se implementa con `cirq.H`, por lo que $\boxed{b = H}$.

9. Question

How many different functions $f : \{0,1\} \to \{0,1\}$ exist?

- 1
- 4

10. Question

The diffusion operator can be written in the form:
(You can choose multiple choices)

Sea $|u\rangle = H^{\otimes n}|0\rangle$. El operador de difusión de Grover puede escribirse habitualmente como

$$D = 2|u\rangle\langle u| - I.$$

Otra forma equivalente (diferente sólo por una fase global -1) es

$$I - 2|u\rangle\langle u| = -D.$$

Analicemos las opciones dadas:

1. $I - 2|u\rangle\langle u|$. **Correcta.** Es una forma válida (igual a $-D$); físicamente equivalente a D porque difiere sólo por una fase global.
2. $I - 2(H^{\otimes n}|0\rangle\langle 0|H^{\otimes n})$. **Correcta.** Puesto que $|u\rangle = H^{\otimes n}|0\rangle$, esta es la misma expresión que (1).
3. $2(|u\rangle\langle u| - I)$. **Incorrecta.** Esto equivale a $2|u\rangle\langle u| - 2I$, que no es D ni su negativo (los factores no coinciden).
4. $2|u\rangle\langle u| - I$. **Correcta.** Es la forma estándar D.
5. $I - 2(H^{\otimes n}|1\rangle\langle 1|H^{\otimes n})$. **Incorrecta.** Conjugar $|1\rangle\langle 1|$ por Hadamards no produce el proyecto sobre $|u\rangle$.

Selección final (correctas):

$$\boxed{\text{(1), (2) y (4).}}$$

Nota: (1) y (4) difieren sólo por un signo global -1, que no afecta probabilidades de medida, por eso ambas son aceptables en la práctica.

11. Question

How does the oracle in Simon's algorithm operate on its input qubits?

- It performs a bitwise AND operation between the input qubits and the secret string s.
- It entangles each input qubit with a corresponding output qubit.
- It flips the sign of the input state corresponding to the secret string s.
- **It maps input qubits to output qubits based on the secret string s using XOR.**

13. Question

In Simon's algorithm, the quantum part of the computation helps to find:

- The minimum number of qubits needed to encode s.
- The period of the function used in the algorithm.
- **A set of linear equations whose solution reveals s.**
- The exact secret string s directly.

14. Question

Let $f(x)$ be a 2-bit to 1-bit function. Then U_f maps the state $|x\rangle|y\rangle$ to state $|x\rangle|y \oplus f(x)\rangle$, where x is a 2-bit string and y a 1-bit string.

Let $f(x)$ be defined as

$$f(00) = 1, \quad f(01) = 0, \quad f(10) = 0, \quad f(11) = 1,$$

If we apply U_f on the state $|00\rangle|-\rangle$, what is the output state?

We have:

$$f(00) = 1, \quad f(01) = 0, \quad f(10) = 0, \quad f(11) = 1.$$

and the oracle acts as

$$U_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \oplus f(x)\right\rangle .$$

We apply U_f to the input

$$\left|00\right\rangle \left|-\right\rangle , \quad \text{where} \quad \left|-\right\rangle = \frac{\left|0\right\rangle - \left|1\right\rangle}{\sqrt{2}}.$$

Step 1 — Apply U_f
Because the oracle flips the second register when $f(x) = 1$, we have

$$U_f \left|x\right\rangle \left|-\right\rangle = (-1)^{f(x)} \left|x\right\rangle \left|-\right\rangle .$$

That's because

$$\left|y \oplus f(x)\right\rangle = \begin{cases} \left|y\right\rangle , & f(x) = 0, \\ X \left|y\right\rangle , & f(x) = 1, \end{cases} \quad \Rightarrow \quad X \left|-\right\rangle = - \left|-\right\rangle .$$

Hence each component acquires a phase $(-1)^{f(x)}$.

Step 2 — Apply to $|x\rangle = |00\rangle$
We only have the component $|00\rangle|-\rangle$. Since $f(00) = 1$,

$$(-1)^{f(00)} = (-1)^1 = -1.$$

So

$$U_f \left|00\right\rangle \left|-\right\rangle = - \left|00\right\rangle \left|-\right\rangle .$$

Final Answer:

$$\boxed{U_f \left|00\right\rangle \left|-\right\rangle = - \left|00\right\rangle \left|-\right\rangle .}$$

15. Question

We use Grover's search algorithm on a list with 512 elements, and there are three marked elements.
When representing the computation on the unit circle, what is the angle of the initial state (immediately after applying Hadamard operators) in degrees?

Remark that π radians is 180 degrees.

Dado que $N = 512$ y $M = 3$, el ángulo inicial θ cumple

$$\sin\theta = \sqrt{\frac{M}{N}} = \sqrt{\frac{3}{512}}.$$

Por tanto,

$$\theta = \arcsin\left(\sqrt{\frac{3}{512}}\right) \approx 0{,}0766 \text{ rad} = 4{,}39^\circ.$$

$$\boxed{\theta = 4{,}39^\circ.}$$

16. Question

Suppose that we have the following quantum state:

$$\left(\frac{1}{\sqrt{2}}(-1)^{f(0)}|0\rangle + \frac{1}{\sqrt{2}}(-1)^{f(1)}|1\rangle\right)|-\rangle$$

If f is a balanced function, which one(s) of the followings can be the state of the first qubit?

$|-\rangle$

Figura 23: Also the sign(-) one

17. Question

Complete the code so that the function oracle implements the function $f(0) = 0, f(1) = 1$. Note that qubit 0 is the input and qubit 1 is the output.

```
def oracle():
    circuit = QuantumCircuit(2)
    # Your code here
    return circuit
```

Queremos que el oráculo actúe como

$$U_f\,|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle,$$

con

$$f(0) = 0, \quad f(1) = 1.$$

Esto significa:

$$|0\rangle|y\rangle \to |0\rangle|y\rangle,$$
$$|1\rangle|y\rangle \to |1\rangle|y \oplus 1\rangle.$$

Es decir:

Control	Objetivo (antes)	Objetivo (después)
0	0	0
0	1	1
1	0	1
1	1	0

Figura 24: Esto es lo que significa $f(input) = output$.

Por tanto, cuando el qubit de entrada (`qubit 0`) es $|1\rangle$, debemos aplicar una compuerta `X` al qubit de salida (`qubit 1`). Esto se implementa con una compuerta **CNOT** controlada por el qubit 0 y con objetivo el qubit 1.

Código en Qiskit

```
def oracle():
    circuit = QuantumCircuit(2)
    circuit.cx(0, 1)   # aplica f(0)=0, f(1)=1
    return circuit
```

Resultado

El oráculo aplica:

$$|x\rangle|y\rangle \longmapsto |x\rangle|y \oplus x\rangle,$$

que corresponde exactamente a $f(x) = x$.

18. Question

Select the true statements.

- [ ] Bernstein-Vazirani algorithm provides an exponential separation with respect to an oracle between probabilistic and quantum computation.
- [x] Deustch-Jozsa and Bernstein-Vazirani algorithms have the same circuit implementation.
- [x] Deustch Algorithm makes only a single query to the oracle.
- [ ] In Bersntesin-Vazirani algorithm, the problem is to decide whether a given function is constant or balanced.

19. Question

Select the functions which are constant.

- [x] $f(0) = 1, f(1) = 1$
- [x] $f(0) = 0, f(1) = 0$

20. Question

Find the secret string s if

$$f(00) = 0, f(01) = 1, f(10) = 1, f(11) = 0$$

Si etiquetamos los bits de entrada como $x = (x_1, x_0)$ (bit más significativo x_1), comprobamos que

$$f(x) = x_1 \oplus x_0,$$

porque:

x_1x_0	$x_1 \oplus x_0$
00	0
01	1
10	1
11	0

Esto es una función lineal sobre $\mathbb{F}_2$ del tipo $f(x) = s \cdot x$ con $s = (1, 1)$. También es una función *balanceada* (mitad ceros, mitad unos).

Oráculo cuántico (forma estándar)

El oráculo actúa como

$$U_f \, |x\rangle|y\rangle = |x\rangle \, |y \oplus f(x)\rangle.$$

Dado $f(x) = x_1 \oplus x_0$, se implementa con dos CNOTs (cada CNOT añade x_i al qubit objetivo):

$$U_f = \mathrm{CNOT}(q_0 \to \mathrm{out}) \;\cdot\; \mathrm{CNOT}(q_1 \to \mathrm{out}),$$

donde tomamos la convención habitual $\texttt{q[0]} = x_0$, $\texttt{q[1]} = x_1$ (o ajusta índices si tu convención es distinta).

Código de ejemplo

Qiskit

```
from qiskit import QuantumCircuit

def oracle():
    circuit = QuantumCircuit(3)  # q0,q1,input; q2 = target (output)
    circuit.cx(0, 2)   # CNOT from x0 to output
    circuit.cx(1, 2)   # CNOT from x1 to output
    return circuit
```

Cirq

```
import cirq

qq = cirq.LineQubit.range(3)  # qq[0]=x0, qq[1]=x1, qq[2]=output
circuit = cirq.Circuit()
circuit.append(cirq.CNOT(qq[0], qq[2]))
circuit.append(cirq.CNOT(qq[1], qq[2]))
```

Observaciones

- Si se usa la versión del oráculo que implementa $|x\rangle|y\rangle \mapsto |x\rangle(-1)^{f(x)}|y\rangle$ (oráculo de fase con ancilla preparada en $|-\rangle$), se puede obtener la fase $(-1)^{x_1 \oplus x_0}$ conjugando apropiadamente.
- En el contexto de Bernstein–Vazirani, este oráculo permite recuperar el vector secreto $s = (1, 1)$ con una sola evaluación cuántica.

0.1. M4: Quantum Computing Platform

Superconducting qubits

Superconducting qubits are a cornerstone of modern quantum computing, widely researched by major technology companies such as IBM, Google, and Rigetti. These qubits are typically made from materials like aluminium or niobium, which exhibit zero electrical resistance at extremely low temperatures. This allows for the creation of highly stable and coherent quantum states using the principle of superconductivity, offering high-speed operations and compatibility with well-established fabrication techniques from the semiconductor industry.

Known for their rapid gate times and high fidelity, these systems operate at millikelvin temperatures, and are maintained by advanced cryogenic systems. These features make them a strong candidate for scalable quantum computing despite challenges related to error rates and cooling requirements.

How They work?

- Superconducting qubits operate using Josephson junctions which consist of two superconducting materials separated by a thin insulating barrier. This configuration enables the formation of quantized energy levels, which can be exploited to define the $|0\rangle$ and $|1\rangle$ states of a qubit. The junction enables the quantum phenomena necessary for qubit operations such that quantum information can be manipulated within these states for performing complex computations.
- Microwave resonators are used to control and read out the state of the qubit. These resonators couple to the qubits and help mediate interactions between them or between a qubit and a measurement device.
- Control lines are used to send precise microwave pulses to manipulate qubit states. These lines allow researchers to perform gate operations, the quantum equivalent of classical logic gates.
- Superconducting qubits must operate at millikelvin temperatures to maintain their superconducting state and minimise thermal noise. Dilution refrigerators achieve these ultra-low temperatures by using helium isotopes.
- Qubit states are measured using dispersive readout techniques, where changes in the resonator frequency indicate the qubit's state. These measurements collapse the qubit's superposition into one of its basis states

Advantages: Rapid gate operations, Scalability potential, Flexibility in design.

Disadvantages: Cooling, decoherence and noise, error correction overhead.

Trapped ion qubits

Trapped ion qubits are among the most stable and accurate qubit platforms available today. These qubits are encoded in the electronic states of individual ions—charged atoms—which are confined and manipulated using electromagnetic fields inside a vacuum chamber. Companies such as IonQ, Quantinuum, and Alpine Quantum Technologies are leading the development of trapped ion quantum computers due to their exceptional coherence times and high-fidelity gate operations.

How They work?

- Ion trapping: Ions are confined in a vacuum using electromagnetic fields generated by a device called an ion trap. These traps include Paul traps and Penning traps, which use different configurations of oscillating electric fields to trap ions in a stable position.
- Qubit initialisation: The ions are cooled to their ground state using laser cooling techniques. The ions are used to represent qubits, with each ion's electronic states corresponding to the $|0\rangle$ and $|1\rangle$ states of a qubit, which can be initialised to a specific quantum state.
- Quantum gates: Quantum operations are performed using laser pulses. These pulses manipulate the internal states of the ions and their collective motion, enabling single-qubit and multi-qubit gates.
- Entanglement: Ions can be entangled by coupling their internal states with their shared motion states. This is typically achieved using laser-induced interactions.
- Measurement: The state of each ion is measured using laser-induced fluorescence. The ions emit light when they are in a specific state and this light is detected to read out the qubit states.

Advantages: Long coherence times, High-fidelity gates, All-to-all connectivity.

Disadvantages: Slow gate speeds, scalability limitations, Complex control requirements.

Photonics qubits

Photonic qubits make use of individual photons as carriers of quantum information, leveraging their quantum properties such as polarisation, phase, or path encoding to perform computations. Unlike superconducting qubits, photonic qubits travel freely through optical circuits and fibre networks, making them particularly advantageous for scalable, high-speed, and long-distance quantum communication. Companies like Xanadu, PsiQuantum, and Quandela are leading efforts to develop large-scale photonic quantum processors.

How they work?

- Photon generation: Photons are generated using sources like lasers or spontaneous parametric down-conversion (SPDC) processes that produce single photons or entangled photon pairs. These photons act as qubits, the basic units of quantum information.
- Quantum state preparation: The photons are prepared in specific quantum states using devices called squeezers. Squeezers manipulate the quantum properties of light to create entangled states, which have reduced quantum noise in one property at the expense of increased noise in the conjugate property and are crucial for quantum computations.
- Interference and manipulation: The photons then pass through a network of beam splitters, phase shifters, and interferometers. These components manipulate the paths and phases of the photons, allowing for the implementation of quantum gates and operations. Beam splitters split a beam of light into two or more separate beams. They are used to create superpositions and entanglements. Phase shifters adjust the phase of the photons, which is crucial for implementing quantum gates. Interferometers combine multiple beams of light to create interference patterns, enabling complex quantum operations.
- Detection and measurement: Finally, the photons are detected using highly sensitive single-photon detectors. The measurement outcomes are used to infer the results of the quantum computation.

Advantages: Room temperature, minimal decoherence, scalability and networking potential

Disadvantages: weak $\gamma - \gamma$ interaction, difficult detection, probabilistic gates.

Neutral atom qubits

Neutral atom qubits use individual uncharged atoms, such as Rubidium or Cesium, which are confined and manipulated using optical tweezers—highly focused laser beams that hold the atoms in precise spatial arrangements. Unlike charged ions, neutral atoms experience minimal electrostatic repulsion, allowing them to be densely packed into large, scalable arrays. Companies like Pasqal, QuEra, and Infleqtion are at the forefront of developing neutral atom quantum processors.

How they work?

- Atom trapping: Neutral atoms are trapped using optical tweezers, which are highly focused laser beams. These lasers create potential wells that hold the atoms in place.
- Qubit initialisation: The atoms are cooled to near absolute zero using laser cooling techniques. This reduces their thermal motion, allowing precise control. The qubits are encoded in the internal energy levels of the atoms.
- Quantum gates: Quantum operations are performed using laser pulses. These pulses manipulate the internal states of the atoms, enabling single-qubit and multi-qubit gates. The Rydberg blockade effect is often used to create entanglement between atoms.
- Entanglement: Atoms can be entangled by exciting them to high-energy Rydberg states, where they interact strongly with each other. This interaction is used to perform two-qubit gates.
- Measurement: The state of each atom is measured using fluorescence detection. Lasers excite the atoms, causing them to emit light, which is then detected to read out the qubit states.

Advantages: Scalability, high fidelity gates, flexible qubit connectivity

Disadvantages: Controlled interactions. Complex laser system, error correction and readout.

Silicon Spin qubits

A silicon spin qubit system consists of quantum dots, which are nanoscale regions in a silicon substrate where single electrons are trapped and manipulated. The electron's spin-up and spin-down states serve as the qubit's logical states to encode and process quantum information. Quantum operations are performed by applying radiofrequency or microwave pulses, which control the spin state through electron spin resonance (ESR) or nuclear magnetic resonance (NMR). Companies such as Intel, Quantum Brilliance, and Silicon Quantum Computing are actively developing silicon-based quantum processors due to their compatibility with existing chip fabrication methods and potential for high-density qubit integration.

How they work?

- Qubit initialisation: The qubits are typically electrons or nuclei in silicon. These spins are initialised to a known state using magnetic fields and microwave pulses.
- Quantum gates: Quantum operations are performed using microwave or radiofrequency

pulses that manipulate the spin states. Single-qubit gates are achieved by rotating the spin, while two-qubit gates are implemented through spin-spin interactions.

- Entanglement: Spins can be entangled by coupling them through exchange interactions or by using a shared quantum dot. This allows for the creation of entangled qubit pairs necessary for quantum computations.
- Measurement: The state of each qubit is measured using techniques like spin-to-charge conversion, where the spin state is converted to an electrical signal that can be detected.

Advantages: Seamless integration with existing technology, long coherence times in pure silicon, compact qubit architecture, energy-efficiente operations.

Disadvantages: Precision in qubit coupling, cryogenic cooling, scalability of control electronics.

Nitrogen-vacancy centres qubits

Nitrogen-vacancy (NV) Centre Quantum Computers are a type of quantum computing platform that utilises defects in diamond crystals as qubits. NV centres are defects found in diamond crystals where a nitrogen atom replaces a carbon atom, and an adjacent carbon atom is missing (creating a vacancy). These defects in the diamond lattice form localised electronic states that can be manipulated and measured by exploiting quantum properties such as spin states and optical transitions to perform quantum operations. NV centres are remarkable for their ability to operate at room temperature, high coherence times, and sensitivity to magnetic and electric fields.

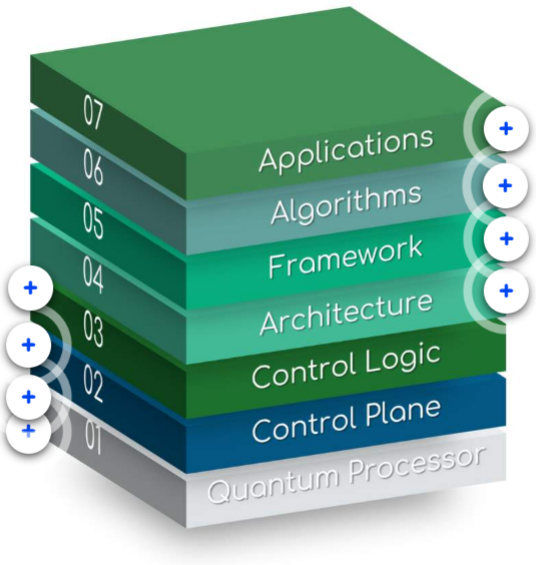
How they work?

- NV centre formation: NV centres are created by replacing a carbon atom in the diamond lattice with a nitrogen atom and creating an adjacent vacancy (a missing carbon atom). This defect forms the NV centre, which can exist in different charge states, typically NV and NV.
- Qubit initialisation: The NV centre's electronic spin states are used as qubits. These states can be initialized using laser light, which prepares the NV centre in a specific quantum state.
- Quantum gates: Quantum operations are performed using microwave and laser pulses. These pulses manipulate the spin states of the NV centres, enabling single-qubit and multi-qubit gates.
- Entanglement: NV centres can be entangled with each other through their spin states. This is often achieved using optical or microwave interactions, allowing for the creation of entangled qubit pairs.
- Measurement: The state of the NV centre is measured using optically detected magnetic resonance (ODMR). When the NV centre is illuminated with a laser, it emits fluorescence that depends on its spin state. This fluorescence is detected to read out the qubit state.

Advantages: Room temperature, long coherences times, optical readout and control, hybrid quantum integration.

Disadvanatges: Qubit scalability, low qubit density, readout efficiency.

Quantum Software



Hardware Layers (Processors & Control Systems)

At the base of the FTQC (Fault-Tolerant Quantum Computing) stack lie the quantum hardware layers, composed of quantum processors (qubits) and control systems that manipulate them. These systems can be based on superconducting qubits, trapped ions, or other quantum technologies.

Quantum Control Systems

According to McKinsey, the control plane layers are responsible for controlling and measuring quantum states, interfacing directly with quantum hardware. They include systems that initialise, manipulate, and read out qubits. Control electronics ensure precise timing and execution of quantum gates, while readout systems measure the quantum state after computation, collapsing qubits into classical bits.

Although all are founded on similar principles of quantum control, their hardware implementations vary according to the qubit technology used. The FTQC architecture focuses on reducing errors through robust error-correction techniques, introducing fault-tolerant quantum gates and circuits that support error detection and correction, enabling scalability.

Software Layers (Processing & Optimisation)

The software layers ensure that quantum programs can be efficiently designed, optimised, and executed on quantum hardware. Quantum error-correction methods—such as surface codes and concatenated codes—protect quantum information from decoherence and other errors. FTQC relies on **logical qubits**, encoded from multiple physical qubits to improve resilience.

Software tools include compilers, middleware, and hybrid integration resources that enable efficient interaction between programmers and quantum hardware. At higher levels, quantum algorithms drive practical applications across industries.

The quantum software layers bridge the gap between hardware and applications, offering frameworks for developing and executing quantum algorithms. Their main components are as follows:

1. Software Development Kits (SDKs)

SDKs provide developers with essential tools, libraries, and APIs for creating and testing quantum algorithms. They simplify the process of writing quantum code, integrating it with classical systems, and typically include simulators, programming languages, and visualisation tools for designing, executing, and debugging quantum circuits.

2. Programming Languages

Quantum programming languages define the structure and execution of quantum algorithms. Popular languages enabling access to real quantum computers include Python, Julia, C++, JSON, and JavaScript—all open-source. Specialised languages such as Q#, Quipper, and pyQML offer constructs specific to quantum computing.

Quantum assembly languages (QASM), particularly OpenQASM, are widely used for circuit design, combining features of C and Assembly. Classical languages often use libraries that extend QASM functionalities.

As quantum computers evolve, enterprise systems will increasingly integrate quantum resources, either via on-premises processors or through cloud-based *Quantum-as-a-Service* (QaaS) models.

3. Error Correction Software

These systems implement quantum error-correction codes and mitigation strategies to preserve computation integrity. Due to their fragile nature, quantum systems are prone to several errors:

- **Decoherence:** Interaction with the environment leading to information loss.
- **Gate Errors:** Imperfect gate operations.
- **Measurement Errors:** Inaccuracies during readout.

Two main decoherence mechanisms are:

- **Amplitude Damping:** Energy loss from qubits (e.g., $|1\rangle \rightarrow |0\rangle$).
- **Phase Damping:** Loss of phase information without energy change.

Error-correction software encodes logical qubits using multiple physical qubits, allowing error detection and correction without collapsing the quantum state, ensuring accurate and efficient computation.

4. Quantum Error-Correcting Codes

Quantum error-correcting codes form the foundation for maintaining computation integrity in FTQC systems. They enable detection and correction of both bit-flip and phase-flip errors, ensuring fault-tolerant operation.

5. Compilers and Optimisers

Quantum compilers translate high-level program gates into universal physical gates and ultimately into control pulses (a process called *transpiling*). These compilers act as optimisers, reducing the number of quantum gates through various optimisation strategies.

Transpilation scales poorly with increasing qubits and gates, requiring sophisticated optimisation methods. Optimisation also involves hardware–software co-design to adapt programs to specific architectures and mitigate system limitations.

6. Cloud Computing Platforms

Leading cloud providers now offer public quantum services with limited processing power. Cloud-based quantum computing platforms provide a user-friendly interface to access quantum capabilities remotely, removing the need for local hardware.

These platforms connect quantum algorithms with real quantum processors, managing execution and resource allocation. They offer cost-effective and scalable access to both simulated and real quantum environments, fostering innovation and collaboration in research and industry. Users can develop and optimise algorithms, experiment with simulations, and prepare for fully fault-tolerant quantum systems.

Cloud platforms thus promote accessibility, scalability, and flexibility, allowing users to align quantum resource use with their project needs and budgets.

Hybrid Quantum Computing and Quantum-Inspired Solutions

A critical aspect of hybrid quantum computing is **quantum-inspired solutions**, which involve developing and optimising classical techniques that mimic or complement quantum computational principles. These methods do not require actual quantum hardware, yet they can offer significant computational advantages for complex optimisation, machine learning, and simulation problems.

Quantum-inspired approaches advance computational capabilities and prepare the ground for the future integration of quantum technologies. Below, we discuss some of the main hybrid quantum computing platforms currently in use.

Quantum Simulators and Emulators

Quantum simulation replicates the behaviour of quantum systems using another controllable quantum system. Its main goal is not computation but **emulation**, allowing researchers to explore complex physical and chemical systems, such as chemical reactions or condensed matter phenomena.

Quantum Simulators

Quantum simulators come in two primary forms:

- **Digital Simulators:** Use discrete qubits and quantum algorithms to emulate quantum systems.
- **Analog Simulators:** Exploit the natural quantum behaviour of one system to imitate another, providing a more direct emulation.

Quantum simulators are widely applied in finance, energy, chemistry, and aerospace. They enable portfolio optimisation, energy grid modelling, molecular structure and reaction analysis, and materials design. In condensed matter physics, they help study inherently quantum phenomena like superconductivity and magnetism.

Quantum Annealers

Quantum annealers are special-purpose quantum computers designed to solve optimisation problems. They use quantum fluctuations to explore large solution spaces efficiently and identify optimal solutions.

- **Purpose and Design:** Tailored for optimisation rather than general-purpose computing.
- **Algorithmic Approach:** Employs quantum annealing processes to reach global minima in complex landscapes.
- **Hardware Implementation:** Built to exploit adiabatic evolution for problem-solving.

In contrast, **gate-based quantum computers** are general-purpose machines that use quantum gates to execute algorithms like Shor's (for factoring) or Grover's (for search).

High-Performance Computing (HPC) Centres

High-Performance Computing (HPC) refers to systems composed of powerful, parallel processors used to solve complex problems requiring massive computation and memory. HPC systems consist of:

- **Compute Nodes:** Multi-core processors with high-speed interconnects and large memory.
- **Shared Storage:** Centralised access to large datasets and applications.
- **Accelerators:** GPUs and FPGAs for specialised computational tasks.

While HPC is a mature technology, quantum computing remains experimental. However, the two can **complement each other**. HPC can simulate quantum systems and assist in developing algorithms before deployment on quantum hardware.

Advantages of Quantum Computers in HPC Centres

- **Accelerating Complex Calculations:** Quantum processors can drastically speed up specific computational tasks.
- **Enhancing Quantum Machine Learning (QML):** Hybrid models leverage both HPC and quantum computing for advanced pattern recognition.
- **Strengthening Existing HPC Infrastructure:** Integrating quantum resources expands computational capacity.
- **Bridging Classical and Quantum Systems:** Ensures seamless interoperability between classical HPC and emerging quantum platforms.

Current Uses of HPC

HPC currently supports applications in weather forecasting, materials science, computational chemistry, artificial intelligence, and complex data analytics.

Implementations of Quantum Computing in HPC Centres

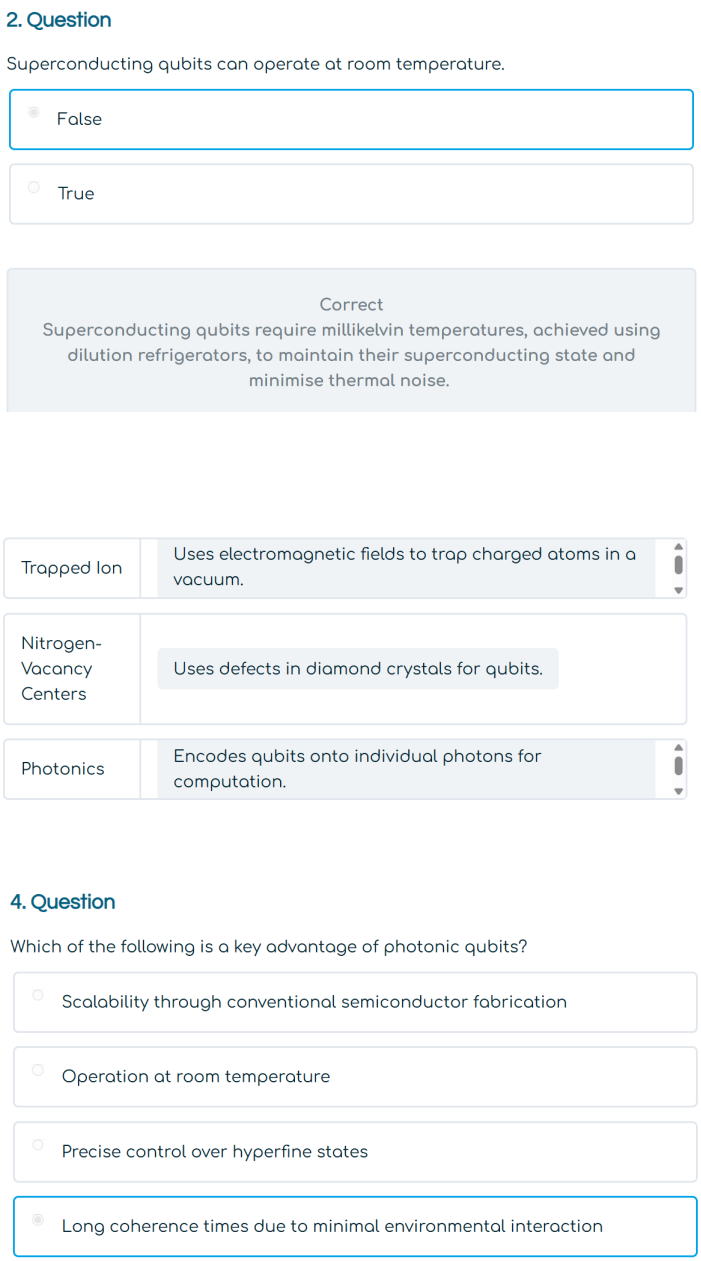
1. **Quantum Computers Integrated into HPC Centres:** HPC centres increasingly offer cloud-based access to quantum computers. Projects such as the *High-Performance Computer – Quantum Simulator hybrid (HPCQS)* integrate quantum simulators with existing HPC infrastructures, providing federated, cloud-accessible environments.
2. **AI Optimising Workloads:** Artificial Intelligence (AI) is used to dynamically allocate HPC resources based on workload requirements. AI algorithms predict computational demands, optimise performance, and reduce energy consumption through intelligent scheduling and cooling strategies.
3. **Edge Computing:** As cloud computing grows, the data influx from IoT and edge devices can overload central systems. Edge computing processes data near its source, reducing latency and bandwidth use. This complements HPC by performing preliminary data processing before transferring essential information to centralised systems, optimising overall performance.

Future of HPC Centres

Future HPC environments will likely be **hybrid systems** that integrate classical HPC, quantum computing, artificial intelligence, and edge computing. These systems will provide scalable, flexible, and powerful computational frameworks to address diverse industrial and scientific challenges.

HPC centres must evolve to integrate new quantum hardware, deploy AI-driven optimisation tools, and support distributed edge infrastructures, ensuring readiness for the next generation of hybrid quantum-classical computing.

Test del Tema 4



5. Question

Neutral atom qubits rely on charged ions trapped in electromagnetic fields.

- True
- False

Correct
Neutral atom qubits rely on the hyperfine splitting of atoms caused by magnetic fields, not charged ions or electromagnetic fields.

6. Question

What is the primary use of Josephson junctions in superconducting qubits?

- To control qubit interactions
- To measure the qubit state
- To provide ultra-low temperature environments
- To enable quantum phenomena like tunneling and superposition

IonQ	Trapped Ion
Xanadu	Photonic
Google Quantum AI	Superconducting
Quantum Brilliance	Silicon Spin/Quantum Dots
Pasqal	Neutral atom

8. Question

What is the main purpose of quantum error correction techniques?

- To improve the speed of quantum computations
- To detect and correct errors in quantum computations
- To allow qubits to operate at room temperature
- To simplify quantum circuit designs

9. Question

Error correction codes like the surface code are used to correct both bit-flip and phase-flip errors in quantum systems.

- False
- True

Correct
The surface code is a robust error-correction code designed to handle both bit-flip and phase-flip errors, making it highly suitable for fault-tolerant quantum computing.

Compiler	Transforms quantum programs into physical gate operations
Error Correction Software	Detects and corrects errors in quantum computations
SDKs	Provides tools for writing, simulating, and running quantum algorithms

11. Question

Which of the following is NOT a quantum error-correcting code?

- Steane Code
- Surface Code
- Shor Code
- Quantum Fourier Code

12. Question

What is a key advantage of cloud-based quantum computing platforms?

- They eliminate the need for qubit error correction.
- They operate quantum computers at room temperature.
- They allow users to access quantum hardware without owning it.
- They guarantee fault-tolerant quantum computations.

13. Question

What is the primary purpose of hybrid quantum computing?

- [x] To combine classical and quantum systems to solve problems efficiently
- [ ] To replace classical computers entirely
- [ ] To allow quantum computers to perform classical tasks
- [ ] To eliminate errors in quantum systems

14. Question

Quantum simulators are primarily used to perform quantum computations.

- [ ] True
- [x] False

Correct
Quantum simulators are designed to emulate quantum systems, allowing researchers to study the behavior of complex quantum phenomena rather than perform general quantum computations.

15. Question

Which feature distinguishes quantum annealing from gate-based quantum computing?

- [ ] It requires precise control over qubit operations
- [ ] It uses quantum gates to manipulate qubits
- [x] It specialises in solving optimisation problems
- [ ] It supports Shor's algorithm for factoring

16. Question

High-performance computing (HPC) systems often use GPUs and FPGAs to accelerate computations.

- [ ] False
- [x] True

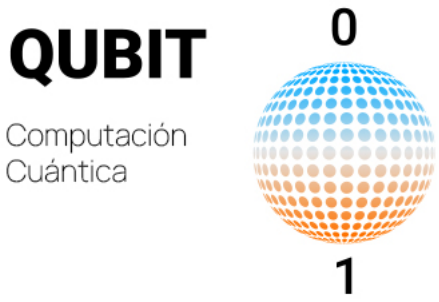
Correct
HPC systems incorporate specialized hardware accelerators like GPUs and FPGAs to enhance the performance of specific computations.

2. Question

Superconducting qubits can operate at room temperature.

- False
- True

Correct
Superconducting qubits require millikelvin temperatures, achieved using dilution refrigerators, to maintain their superconducting state and minimise thermal noise.



Computación Cuántica

Curso Qureca

Año: 2025/26

M5: Subroutines (Quantum Fourier Transform and Quantum Phase Estimation)

Complex numbers in Phyton:

```
z1 = 3+2j
print(z1)
z2 = complex(3,2)
print(z2)
z3 = 5j
print(z3)
z4 = complex(0,5)
print(z4)
z = 3+2j
print('real part of 3+2j:',z.real)
print('imaginary part of 3+2j:',z.imag)
print('conjugate of 3+2j:',z.conjugate())
print('absolute value of 3+2j:',abs(z))
print('(3+2j)^3:',pow(z,3))
```

Discrete Fourier transformation

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi i \cdot jk}{N}} x_j \rightarrow \begin{pmatrix} 3 \\ 2 \end{pmatrix} \equiv \begin{pmatrix} \frac{3}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix}$$

Quantum Fourier Transform

What makes Fourier Transform special is that it can be computed faster on a quantum computer than on a classical computer. In this notebook, we will learn about the quantum analogue of Discrete Fourier Transform.

Definition of DFT

Let's recall the definition for DFT. DFT of the vector

$$x = (x_0, x_1, \ldots, x_{N-1})^T$$

is the complex vector

$$y = (y_0, y_1, \ldots, y_{N-1})^T$$

where

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i jk/N} x_j.$$

Quantum Fourier Transform (QFT)

Now suppose that we have an $N = 2^n$-dimensional quantum state vector

$$x = (x_0, x_1, \ldots, x_{N-1})^T$$

representing the state

$$|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle .$$

Quantum Fourier Transform (QFT) of state $|\psi\rangle$ is given by

$$|\phi\rangle = \sum_{k=0}^{N-1} y_k |k\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} e^{2\pi ijk/N} x_j |k\rangle ,$$

where y_k is defined as above.

QFT of Basis States

For a basis state $|j\rangle$ represented by $x_j = 1$ and all other entries 0,

$$\mathrm{QFT}(|j\rangle) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{jk} |k\rangle ,$$

where $\omega = e^{2\pi i/N}$.

QFT Matrix

QFT is a linear transformation represented by an $N \times N$ matrix:

$$\mathrm{QFT} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix} .$$

Task 1 (on paper)

Apply QFT to the basis state $|10\rangle$ and find the new quantum state.

Solution: We have $n = 2$ qubits and $N = 4$. Then $\omega = e^{2\pi i/4} = i$. Since $|10\rangle$ corresponds to $j = 2$,

$$\mathrm{QFT}\,|10\rangle = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle).$$

QFT Operator Properties

QFT is a unitary operator. The $(j+1)$-th row is

$$\frac{1}{\sqrt{N}}(1, \omega^j, \omega^{2j}, \ldots, \omega^{j(N-1)}),$$

and all rows (and columns) form an orthonormal set.

Inverse QFT

The inverse QFT is also unitary and equals $\mathrm{QFT}^\dagger$:

$$\mathrm{QFT}^{-1} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(N-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-2(N-1)} & \cdots & \omega^{-(N-1)^2} \end{pmatrix} .$$

Linear Shift Property

A linear shift of a state vector causes a relative phase shift of its QFT, and vice versa.

Circuit Implementation of QFT

For one qubit, QFT is equivalent to applying a Hadamard gate:

$$\mathrm{QFT}\,|\psi\rangle = \frac{1}{\sqrt{2}} \left[(\alpha + \beta) |0\rangle + (\alpha - \beta) |1\rangle \right] .$$

Multi-Qubit QFT

For n qubits, let $N = 2^n$ and

$$|j\rangle = |j_1 j_2 \cdots j_n\rangle = |j_1\rangle \otimes |j_2\rangle \otimes \cdots \otimes |j_n\rangle ,$$

with $j = j_1 2^{n-1} + j_2 2^{n-2} + \cdots + j_n 2^0$.

Then:

$$\mathrm{QFT}\,|j\rangle = \frac{1}{\sqrt{N}} \bigotimes_{t=1}^{n} \left(|0\rangle + e^{2\pi i (0.j_{n-t+1} \cdots j_n)}\,|1\rangle\right).$$

Circuit Design

To realize this, we:

1. Apply a Hadamard gate to the first qubit.
2. Apply controlled phase gates $R_m = \mathrm{diag}(1, e^{2\pi i/2^m})$ from later qubits.
3. Repeat for each qubit, applying one fewer control each time.
4. Finally, apply SWAP gates to reverse the qubit order.

QFT in Cirq

Cirq provides a controlled phase gate:

$$\mathrm{CZPowGate}(t) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\pi i t} \end{pmatrix}.$$

By setting $t = 2/2^m$, we create the CR_m gate.

Task 2

Write a method `myQFT` that takes a list of qubits and returns a circuit implementing QFT.

Code:

```
!pip install cirq
import cirq
from cirq import H, SWAP
from cirq.circuits import InsertStrategy
from math import pi

def myQFT(qubits):
    circuit = cirq.Circuit()
    n = len(qubits)
    for i in range(n):
        circuit.append(H(qubits[i]), strategy=InsertStrategy.NEW)
        phase_divisor = 4
        for j in range(i+1, n):
            circuit.append(cirq.CZPowGate(exponent=2/phase_divisor).on(qubits[j], qubi
            phase_divisor *= 2
    for j in range(n//2):
        circuit.append(SWAP.on(qubits[j], qubits[n-j-1]), strategy=InsertStrategy.NEW)
    return circuit
```

```
qubits = cirq.LineQubit.range(3)
my_circuit = myQFT(qubits)
print("=========== Three qubits ===========")
print(my_circuit)

qubits = cirq.LineQubit.range(4)
my_circuit = myQFT(qubits)
print(my_circuit)
```

*Phase Estimation

Phase Estimation

In this section we will talk about an application of Quantum Fourier Transform, which will lead us in the way to Shor's Algorithm.

For a given matrix A, a nonzero vector v is called its eigenvector if there is a scalar value λ called eigenvalue satisfying

$$A \cdot v = \lambda \cdot v.$$

Unitary matrices are length-preserving. Therefore, their eigenvalues must have magnitude 1, and so they are of the form $e^{2\pi i\phi}$ for some $\phi \in [0, 1)$.

Our goal is, for a given unitary matrix U and its eigenvector $|u\rangle$, to estimate the phase ϕ of the corresponding eigenvalue $e^{2\pi i\phi}$.

The matrix U^k is the k-th power of U. As $U|u\rangle = e^{2\pi i\phi}|u\rangle$, we have

$$U^k|u\rangle = (e^{2\pi i\phi})^k|u\rangle = e^{2\pi i\phi k}|u\rangle.$$

Task 1 (on paper)

Show that $|-\rangle$ and $|+\rangle$ are eigenvectors for the X operator with eigenvalues -1 and 1 respectively.

Solution

$$X|-\rangle = X\frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{|1\rangle - |0\rangle}{\sqrt{2}} = -\frac{|0\rangle - |1\rangle}{\sqrt{2}} = -|-\rangle$$

$$X|+\rangle = X\frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{|1\rangle + |0\rangle}{\sqrt{2}} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle$$

Hence, $|-\rangle$ and $|+\rangle$ are eigenvectors of X operator with eigenvalues -1 and 1 respectively.

Controlled-U Operator

Let U be a unitary operator with eigenstate $|\psi\rangle$ and eigenvalue $e^{2\pi i\phi}$ such that $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$.

In the first case, when the control qubit is $|0\rangle$:

$$CU(|0\rangle|\psi\rangle) \to |0\rangle|\psi\rangle.$$

In the second case, when the control qubit is $|1\rangle$:

$$CU(|1\rangle|\psi\rangle) \to e^{2\pi i\phi}|1\rangle|\psi\rangle.$$

Thus, the CU operator puts a phase of $e^{2\pi i\phi}$ in front of the state $|1\rangle$ when the control qubit is in superposition.

$$CU\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}|\psi\rangle\right) = \frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle + e^{2\pi i\phi}|1\rangle|\psi\rangle) = \left(\frac{|0\rangle + e^{2\pi i\phi}|1\rangle}{\sqrt{2}}\right)|\psi\rangle.$$

Hence, for an arbitrary state $\alpha|0\rangle|\psi\rangle + \beta|1\rangle|\psi\rangle$,

$$CU \longrightarrow \alpha|0\rangle|\psi\rangle + e^{2\pi i\phi}\beta|1\rangle|\psi\rangle = (\alpha|0\rangle + e^{2\pi i\phi}\beta|1\rangle)|\psi\rangle.$$

Task 2

Consider the quantum state (where $x = 0$ or 1):

$$\frac{|0\rangle + (-1)^x|1\rangle}{\sqrt{2}}.$$

How can you find out the value of x by applying a single operator?

Solution

If our aim is to find out x, applying a Hadamard gate will determine whether $x = 0$ or $x = 1$. If the outcome is $|0\rangle$, then $x = 0$. If the outcome is $|1\rangle$, then $x = 1$.

Estimating Eigenvalues of the X Operator

Recall that $|+\rangle$ and $|-\rangle$ are eigenstates of X. If we apply CX operator to the state

$$CX\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}|\psi\rangle\right) \to \frac{|0\rangle + (-1)^x|1\rangle}{\sqrt{2}}|\psi\rangle,$$

then, by Task 2, we can find the value of x.

This is a special case of the phase estimation problem where we estimate eigenvalues of X. Here, ϕ is either 0 or $1/2$, which can be determined with a Hadamard gate.

Algorithm

We use two registers:

- The first register has t qubits initialized to $|0\rangle^{\otimes t}$.
- The second register stores $|\psi\rangle$.

We apply a series of controlled-U^{2^j} operations for $j \in \{0, \ldots, t-1\}$.

Hadamard Operators

First, we apply a Hadamard operator to each qubit on the first register:

$$\frac{1}{2^{t/2}}\left((|0\rangle + |1\rangle)^{\otimes t}\right)|\psi\rangle.$$

Controlled-U^{2^j} Operators

For each j, apply CU^{2^j} to the second register, controlled by qubit q_{t-j} of the first register. After applying all, the state becomes

$$\frac{1}{2^{t/2}}\bigotimes_{k=1}^{t}(|0\rangle + e^{2\pi i\phi/2^{k-1}}|1\rangle)|\psi\rangle.$$

Inverse QFT

After discarding the second register, the state of the first register is:

$$|u\rangle = \frac{1}{2^{t/2}}\bigotimes_{k=1}^{t}(|0\rangle + e^{2\pi i\phi 2^{t-k}}|1\rangle).$$

This is the same as the state obtained by $QFT|j\rangle$. Hence, we can estimate ϕ by applying $QFT^\dagger$ to $|u\rangle$ and measuring.

Circuit

The phase estimation circuit uses $O(t^2)$ gates and t controlled-U^{2^j} operations.

Outcomes

If there exists $|j\rangle$ such that $j = N\phi$, then the outcome is $|j\rangle$ with probability 1 and $\phi = j/N$. Otherwise, $N\phi$ lies between integers j and $j+1$, and measurement gives either $|j\rangle$ or $|j+1\rangle$ with high probability.

To achieve accuracy of m bits with success probability at least $1-\epsilon$,

$$t = m + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil.$$

Example: Single Qubit Unitary Operator

We consider a single qubit unitary operator U that has eigenvector $|1\rangle$ and eigenvalue $e^{2\pi i\phi}$ where $\phi = \frac{5}{16}$.

The gate `CZPowGate(2*5/16)` in Cirq is a good candidate for U.

```
!pip install cirq
import cirq
from cirq import CZPowGate

def CU(power, qcontrol, target):
    circuit = cirq.Circuit()
    circuit.append(CZPowGate(exponent = (2*5/16)*(2**power)).on(qcontrol, *target))
    return circuit
```

Task 3

Pick $t = 4$ and implement the phase estimation circuit to estimate ϕ. Use your `myInvQFT` method (from the QFT notebook) for $QFT^\dagger$.

Code:

```
import cirq
from cirq import H, SWAP, CZPowGate
from cirq.circuits import InsertStrategy

def CU(power, qcontrol, target):
    circuit = cirq.Circuit()
    circuit.append(CZPowGate(exponent = (2*5/16)*(2**power)).on(qcontrol, *target))
    return circuit

def myInvQFT(qubits):
    circuit = cirq.Circuit()
    n = len(qubits)
    for j in range(n//2):
        circuit.append(SWAP.on(qubits[j], qubits[n-j-1]), strategy = InsertStrategy.NE
    for i in range(n-1, -1, -1):
        phase_divisor = 2**(n-i)
        for j in range(n-1, i, -1):
            circuit.append(CZPowGate(exponent = -2/phase_divisor).on(qubits[j], qubits
                           strategy = InsertStrategy.NEW)
            phase_divisor = phase_divisor / 2
        circuit.append(H(qubits[i]), strategy = InsertStrategy.NEW)
    return circuit

import cirq
```

```
from cirq import X, measure

circuit = cirq.Circuit()
t = 4
n = 1
control = [cirq.LineQubit(i) for i in range(t)]
target = [cirq.LineQubit(i) for i in range(t, t+n)]
circuit.append(cirq.H.on_each(control))
circuit.append(X.on_each(target))

for j in range(t):
    circuit += CU(j, control[t-j-1], target)

print(circuit)
circuit += myInvQFT(control)
circuit.append(measure(*control, key='result'))

sim = cirq.Simulator()
results_sim = sim.simulate(circuit)
print(results_sim.dirac_notation())

samples = sim.run(circuit, repetitions=1000)
print(samples.histogram(key='result'))

def bitstring(bits):
    return "".join(str(int(b)) for b in bits)

print(samples.histogram(key='result', fold_func=bitstring))
```

Result:

$$|01011\rangle \quad \Rightarrow \quad 5 \quad \Rightarrow \quad \phi = \frac{5}{16}.$$

M6: Advanced quantum algorithms

Variational algorithms

Variational Quantum Algorithms (VQAs) are a class of hybrid quantum-classical algorithms designed to harness the power of quantum computing while addressing the challenges posed by current quantum hardware.

These algorithms are particularly promising for near-term quantum devices, known as Noisy Intermediate-Scale Quantum (NISQ) devices, which are limited by noise, short coherence times, and relatively small numbers of qubits.

At their core, VQAs combine quantum and classical computing resources in an iterative process to solve complex problems.

In this module, we will explore the principles behind VQAs, the role of quantum and classical components, and how these algorithms are implemented on current quantum hardware.

Parametrized Quantum Circuits (PQCs) Parametrized quantum circuits consist of

quantum gates that have tunable parameters, allowing the quantum state to be manipulated during optimization.

In the context of VQAs, PQCs are crucial because they allow the quantum system to be "tunedïn a way that makes the system more suitable for the problem at hand.

Quantum gates with parameters are those whose parameters are typically represented as angles (e.g., $R_X(\theta), R_Y(\theta)$)

The goal is to optimize these parameters to minimize or maximize a specific cost function, which is often tied to the problem being solved (such as finding the ground state energy in quantum chemistry or solving optimization problems).

$$R_x(\theta) = \begin{pmatrix} \cos\left(\frac{\Theta}{2}\right) & -i\sin\left(\frac{\Theta}{2}\right) \\ -i\sin\left(\frac{\Theta}{2}\right) & \cos\left(\frac{\Theta}{2}\right) \end{pmatrix}, \quad R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\Theta}{2}\right) & -\sin\left(\frac{\Theta}{2}\right) \\ \sin\left(\frac{\Theta}{2}\right) & \cos\left(\frac{\Theta}{2}\right) \end{pmatrix}.$$

The initial state is given by $|\Psi_0\rangle = |00\rangle$. After applying $R_x(\theta_0)$ on qubit 0 and $R_x(\theta_1)$ on qubit 1, we get the following:

$$\begin{aligned} |\Psi_1\rangle &= \left(\cos\left(\frac{\theta_1}{2}\right)|0\rangle + \sin\left(\frac{\theta_1}{2}\right)|1\rangle\right) \otimes \left(\cos\left(\frac{\theta_0}{2}\right)|0\rangle - i\sin\left(\frac{\theta_0}{2}\right)|1\rangle\right) \\ &= \cos\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_0}{2}\right)|00\rangle - i\cos\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_0}{2}\right)|01\rangle + \sin\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_0}{2}\right)|10\rangle - i\sin\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_0}{2}\right)|11\rangle \end{aligned}$$

Next, let us apply the CNOT gate.

$$|\Psi_2\rangle = \cos\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_0}{2}\right)|00\rangle - i\sin\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_0}{2}\right)|01\rangle + \sin\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_0}{2}\right)|10\rangle - i\cos\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_0}{2}\right)|11\rangle$$

For example, in Quantum Chemistry applications (such as the Variational Quantum Eigensolver), PQCs are used to model the quantum states of molecules. These circuits prepare trial states that can then be used to estimate the ground state energy of a molecular system.

In Optimization Problems (such as with the Quantum Approximate Optimization Algorithm), PQCs represent potential solutions to optimization problems. Classical optimization techniques adjust the parameters of the quantum circuit iteratively based on the outcomes of quantum measurements, with the goal of finding the optimal solution.

For Machine Learning applications, PQCs can represent the parameterized models in quantum neural networks. These models are trained to perform tasks such as classification, regression, or clustering.

Ansatz

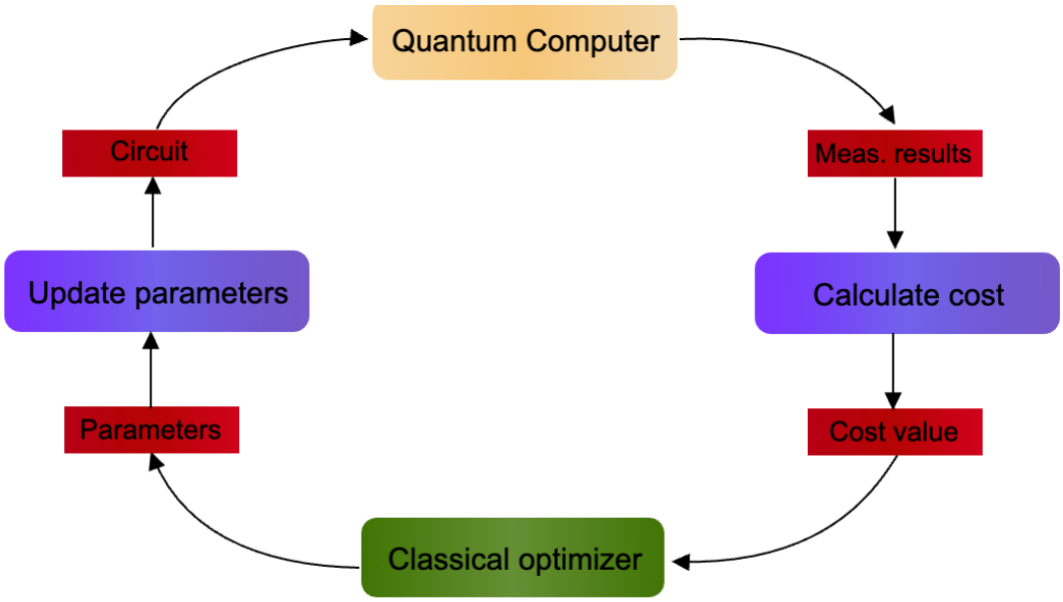
As mentioned above, the parametrized quantum circuits in VQAs often encode potential solutions to problems. Such a trial state, which is often referred to as an ansatz can be seen as a guess for the solution. Construction of an ansatz may be problem inspired, or totally problem-independent, which we will address later on.

One important aspect when choosing an ansatz is expressibility, which can be explained as the possible states that can be obtained by setting different parameters to the ansatz.

For instance, the set of states that are spanned applying $R_X(\theta)$ on a quantum circuit with a single qubit are visualized below on the Bloch sphere.

Optimization in VQAs

In VQAs, optimization of the parameters is essential because the quantum circuit itself doesn't "know"the correct parameters, it only generates a quantum state based on its parameters, and it's the classical optimizer's job to refine those parameters to minimize or maximize a given cost function.



How Classical Optimizers Guide Quantum Circuits In VQAs, a quantum circuit is initialized with some parameters, and the output of the circuit is measured. These measurement results are used to compute a cost function, which tells us how good the current parameters are. A classical optimizer then adjusts the parameters to improve the result.

Cost Function Evaluation The cost function represents what we are trying to minimize or maximize. For example, in quantum chemistry, the cost function might be the energy of a molecule, and in optimization problems, it could represent a score that needs to be minimized. The quantum circuit is run multiple times with different parameters to evaluate how the cost function changes, helping the optimizer decide how to update the parameters.

Variational quantum eigensolver

The Variational Quantum Eigensolver (VQE) is a quantum algorithm designed to approximate (find an upper bound) the ground-state energy of a quantum system. It was proposed by Peruzzo et al. in 2014. In this notebook, we will learn about the basics of it and the motivation behind.

How VQE works? To answer this question, we need to talk about some concepts like ground state, Hamiltonian and variational principle.

What is the ground state? A quantum system can exist in multiple energy states, each associated with a specific energy level. The ground state is the lowest energy state of the system. It is the most stable configuration because, unless external energy is applied, a system will always tend to settle in its ground state.

Ground state energy corresponds to the lowest eigenvalue of the Hamiltonian that represents the system.

What is Hamiltonian? In quantum mechanics, the Hamiltonian is a matrix that represents the total energy of a system. It includes both the kinetic energy (motion of particles) and potential energy (interaction forces). Its eigenvalues correspond to the pos-

sible energy levels of the system which are "allowed", and the corresponding eigenstates are the quantum states that have definite energy values.

The Hamiltonian H must be a Hermitian operator, meaning:

$$H^\dagger = H,$$

where $H^\dagger$ is the conjugate transpose (or adjoint) of the operator H.

This condition ensures that the Hamiltonian has real eigenvalues, which correspond to measurable energy values.

Why are we interested in the ground state energy? The ground-state energy is crucial in many fields, helping predicting chemical reactions, analyzing material properties, and improving drug discovery.

However, calculating the ground-state energy of complex molecules using classical computers is computationally expensive.

Let us now inspect in more detail what a Hamiltonian looks like.

The Pauli Z matrix represents a single-qubit Hamiltonian that can describe the energy associated with the spin of a qubit along the z-axis.

$$\sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

It is Hermitian and its lowest eigenvalue is -1 with the corresponding eigenstate

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Pauli matrices form a complete set of operators for qubits, meaning that any operator acting on a qubit system can be expressed as a linear combination of these Pauli operators. Hence, a general Hamiltonian for a quantum system can be written as a sum of terms called Pauli strings, each of which involves Pauli matrices acting on one or more qubits.

As an example, we can define the following Hamiltonian on two qubits:

$$X_0 \otimes X_1 + Y_0 \otimes Y_1 + Z_0 \otimes Z_1,$$

typically written as $XX + YY + ZZ$. In this example, XX, YY and ZZ are Pauli strings. This Hamiltonian is known as the Heisenberg Hamiltonian. Its matrix representation can be obtained through the matrices of Pauli X, Y, and Z operators.

Here are some other random Hamiltonians:

$$2XIX - YZI, \quad 3XYY + 2ZZI - 4YXY, \quad -XI + YZ + 5XX.$$

The coefficients represent the strength of the interaction between different Pauli operators acting on the qubits, with both positive and negative values indicating different types of interactions (attractive or repulsive) between qubits.

The variational principle For a Hamiltonian H with eigenstates $|\psi_\lambda\rangle$ and corresponding eigenvalues E_λ, the variational principle states that for any normalized trial state $|\psi\rangle$, the expectation value of H satisfies:

$$E_0 \leq \langle\psi| H |\psi\rangle,$$

where E_0 is the ground-state energy (the lowest eigenvalue of H).

Now let us understand why this is the case. We can express the expectation value as follows.

Since any state can be expressed as a linear combination of the eigenstates

$$\langle\psi| H |\psi\rangle = \langle\sum_{\lambda} c_\lambda^* \psi_\lambda| H |\sum_{\mu} c_\mu \psi_\mu\rangle = \sum_{\lambda,\mu} c_\lambda^* c_\mu \langle\psi_\lambda| H |\psi_\mu\rangle .$$

Now since $H |\psi_\lambda\rangle = E_\lambda |\psi_\lambda\rangle$, we get:

$$\langle\psi| H |\psi\rangle = \sum_{\lambda,\mu} c_\lambda^* c_\mu E_\mu \langle\psi_\lambda|\psi_\mu\rangle .$$

Since the eigenstates are orthonormal, all terms vanish except for the case $\mu = \lambda$.

$$\langle\psi| H |\psi\rangle = \sum_{\lambda} |c_\lambda|^2 E_\lambda.$$

Since E_0 is the lowest eigenvalue of H, it follows that:

$$E_0 \sum_{\lambda} |c_\lambda|^2 \leq \sum_{\lambda} |c_\lambda|^2 E_\lambda = \langle\psi| H |\psi\rangle .$$

Since $\sum_\lambda |c_\lambda|^2 = 1$, this simplifies to:

$$E_0 \leq \langle\psi| H |\psi\rangle .$$

What does it signify? This proves that the expectation value of H for any state $|\psi\rangle$ is always greater than or equal to the ground-state energy E_0. In other words, any random guess of a quantum state will provide an upper bound on the ground state energy. This principle forms the basis for the Variational Quantum Eigensolver, which aims to approximate E_0 by optimizing over different trial states.

Implementation

Variational Quantum Eigensolver - Implementation

In the previous notebook, we learnt about the basics of VQE. In this notebook, we will go into implementation details.

How to implement VQE? VQE works by iteratively preparing a trial quantum state with adjustable parameters, measuring its energy on a quantum computer, and using classical optimization to refine the parameters until the lowest possible energy (ground state) is approximated.

In other words, $\langle\psi| H |\psi\rangle$ is computed for different trial states $|\psi\rangle$, whose parameters are adjusted by classical optimization.

Ansatz "Trial quantum state with adjustable parameters" is the ansatz we discussed in the previous notebook. The quality of the ansatz directly affects the efficiency of the algorithm—too simple, and it may not capture the correct solution; too complex, and it may be difficult to optimize.

Common approaches include:

Hardware-Efficient Ansatz: Uses native quantum gates but may suffer from barren plateaus.

Unitary Coupled Cluster (UCC) Ansatz: Popular in quantum chemistry but requires deep circuits.

Problem-Specific Ansatz: Tailored to a given Hamiltonian, often improving efficiency.

Adaptive Ansatz (e.g., ADAPT-VQE): Dynamically constructs an optimal ansatz, reducing circuit depth.

In this notebook, we will not go into details of the mentioned ansatze and keep things simple.

The next thing we need to understand is how to "measure its energy on a quantum computer".

Measurement

You can skip this section if you already know about the measurement concept in quantum computing.

The general way to define a measurement in quantum mechanics with respect to an operator is based on the concept of observable operators and expectation values. Here, we will focus on projective measurements.

An observable in quantum mechanics (such as position, momentum, or spin) is represented by a Hermitian operator. This is because measurements in quantum mechanics must produce real eigenvalues, which are physically interpretable as the possible outcomes of a measurement. Such an observable can be expressed as

$$H = \sum_{\lambda} \lambda P_{\lambda},$$

where P_λ is the projector onto the eigenspace of H with eigenvalue λ. The projectors are constructed as

$$P_\lambda = |\psi_\lambda\rangle \langle\psi_\lambda| ,$$

where $|\psi_\lambda\rangle$ are the normalized eigenvectors of P_λ, which is always possible to find since the observable is Hermitian.

When a measurement is performed, the system collapses to one of the eigenstates of the operator and the outcome of the measurement corresponds to the eigenvalue associated with that eigenstate, with the following probability:

$$p(\lambda) = |\langle\psi_\lambda|\psi\rangle|^2 = \langle\psi|P_\lambda|\psi\rangle .$$

This is known as the Born Rule. The state of the quantum system after the measurement is given by

$$\frac{P_\lambda |\psi\rangle}{\sqrt{p(\lambda)}}.$$

For such observables, expectation value can be expressed as

$$E(H) = \sum_{\lambda} \lambda p(\lambda) = \sum_{\lambda} \langle\psi|P_\lambda|\psi\rangle = \langle\psi|H|\psi\rangle .$$

Let us consider the Z operator. Its eigenvalues are -1 and 1, with the corresponding eigenvectors

$$|\psi_{-1}\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad |\psi_1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

respectively. In this case,

$$P_{-1} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad P_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

Let us take an arbitrary state

$$|\psi\rangle = \frac{1}{\sqrt{3}} |0\rangle + \frac{\sqrt{2}}{\sqrt{3}} |1\rangle .$$

Note that

$$P_{-1} |\psi\rangle = \begin{pmatrix} 0 \\ \frac{\sqrt{2}}{\sqrt{3}} \end{pmatrix}, \quad P_1 |\psi\rangle = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ 0 \end{pmatrix}.$$

□

Therefore,

$$p(-1) = \langle\psi|P_{-1}|\psi\rangle = \frac{2}{3}, \quad p(1) = \langle\psi|P_1|\psi\rangle = \frac{1}{3}.$$

If the outcome is -1, the state of the system after the measurement is given by $|1\rangle$, and if the outcome is 0, it is $|0\rangle$.

The expectation value of Z with respect to $|\psi\rangle$ is

$$\langle\psi| Z |\psi\rangle = \left(\frac{1}{\sqrt{3}}\langle 0| + \frac{\sqrt{2}}{\sqrt{3}}\langle 1|\right)\left(\frac{1}{\sqrt{3}}|0\rangle - \frac{\sqrt{2}}{\sqrt{3}}|1\rangle\right) = \frac{1}{3} - \frac{2}{3} = -\frac{1}{3}.$$

Instead of specifying an observable, the phrase "measurement in computational basis" is often used, where the basis is formed by the corresponding eigenstates of some observable. In quantum computing, in general we measure in the computational basis $\{|0\rangle, |1\rangle\}$, which actually corresponds to measuring with observable Z. When we measure the circuit many times, we can make an estimation on the expectation value of the Z operator, by estimating the probabilities $p(\lambda)$.

How to take expectation value of operators other than Z?

Let us consider X operator for instance.

$$\langle\psi| Z |\psi\rangle = \langle\psi| HXH |\psi\rangle = \langle\psi'| X |\psi'\rangle,$$

where $|\psi'\rangle = H|\psi\rangle$. This observation suggests us a method for measurement in X basis: We can apply H on the quantum state we would like to measure and then perform usual measurement with the Z observable. We can make a similar observation for Y operator:

$$\langle\psi| Z |\psi\rangle = \langle\psi| SHYHS^\dagger |\psi\rangle = \langle\psi''| X |\psi''\rangle,$$

where $|\psi''\rangle = HS^\dagger|\psi\rangle$. Hence, measuring with observable Y, we can first apply $HS^\dagger$ on the circuit and then measure it.

Measurement in VQE

In general, the Hamiltonian whose ground state energy we would like to estimate will consist of the composition of Pauli strings, i.e.,

$$H = H_1 + H_2 + \cdots + H_n.$$

So to compute $\langle\psi| H |\psi\rangle$, we can compute individual terms $\langle\psi| H_i |\psi\rangle$ and take the sum.

Consider $H = 2Y + Z - X$. The Hamiltonian consists of 3 terms, $H_1 = 2Y$, $H_2 = Z$, and $H_3 = -X$. To compute the expectation value of ψ with respect to H, we need to prepare 3 different circuits.

What to do when we have multiple qubits? Suppose that our Hamiltonian is given by Z_0Z_1. The eigenstates of Z_0Z_1 are $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, with the corresponding eigenvalues $1, -1, -1, 1$ respectively. How can we implement this behaviour?

Note that eigenvalues are 1 if two qubits are in the same state, and -1 otherwise.

Let us implement a CNOT gate on the two qubits as

$$\text{CNOT}|q_0\rangle|q_1\rangle = |q_0\rangle|q_0 \oplus q_1\rangle,$$

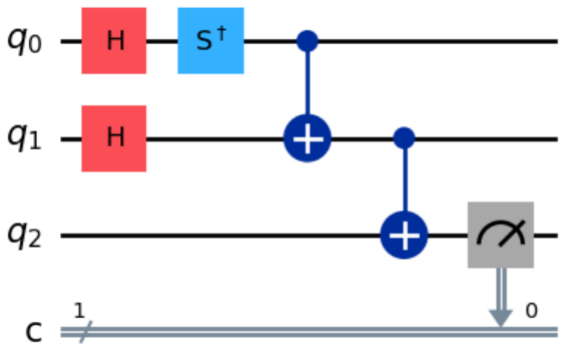
where $|q_0 \oplus q_1\rangle = |0\rangle$ if two qubits have the same state, and $|1\rangle$ otherwise. After applying a CNOT, measurement with Z on qubit q_1 yields us the result we desire.

This idea can be extended to multiple qubits by applying CNOT in a chain structure. Such CNOT chain keeps the parity of 1s in the state and results in state $|1\rangle$ on the final target qubit if the number of 1s in the state is odd, and $|0\rangle$ otherwise. Hence, a

measurement on the final qubit with observable Z coincides with the eigenvalues of the multiqubit observable.

What about other observables like $Y_0X_1Z_2$?

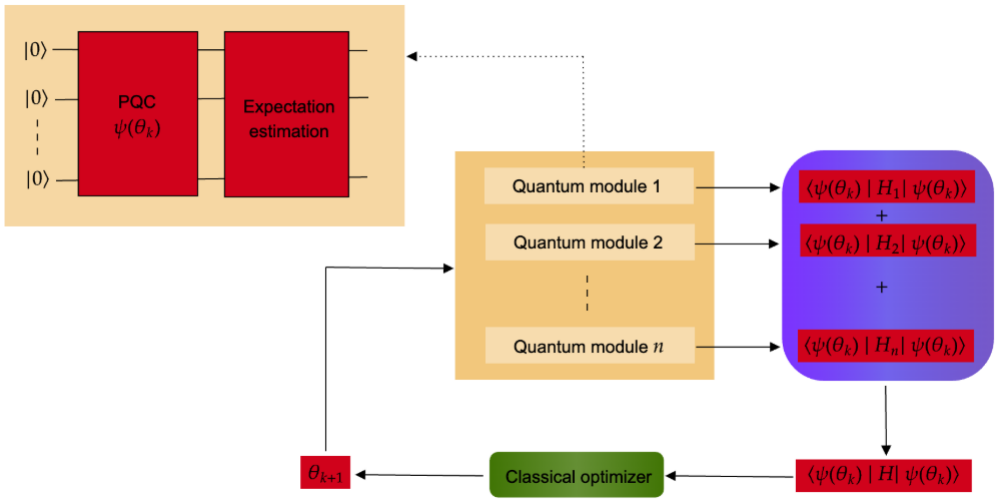
Such observables can be obtained from $Z_0Z_1Z_2$ by making the necessary transformation through H and $HS^\dagger$.



Number of measurements in VQE

A key challenge in VQE is the number of measurements needed to estimate expectation values accurately. The required measurement count scales inversely with the desired accuracy. There are strategies to reduce this, like grouping commuting terms to reduce the number of separate measurements needed.

The algorithm



Number of measurements in VQE Now we can put the pieces together for the VQE algorithm.

We select a parametrized quantum circuit $|\psi(\theta)\rangle$ to serve as our ansatz.

We initialize its parameters $|\psi(\theta_0)\rangle$.

We use a classical optimizer to adjust the parameter θ, by performing measurements to compute $\langle\psi(\theta_0)|\, H\, |\psi(\theta_0)\rangle$, which is the cost function we would like to minimize.

VQE is a hybrid algorithm, where the optimization of the parameters is performed on a classical computer, and the quantum computer is used for calculating the expectation value.

What else about VQE?

VQE is suitable for NISQ (Noisy Intermediate-Scale Quantum) devices due to several

reasons.

Near-Term Applicability: Unlike other quantum algorithms requiring full fault tolerance, VQE can run on noisy intermediate-scale quantum (NISQ) devices.

Variational Flexibility: The algorithm allows for flexible ansatz choices, enabling the adaptation of quantum circuits to hardware constraints.

Hybrid Nature: By offloading optimization tasks to classical computers, VQE mitigates some quantum computational limitations.

Error Mitigation Techniques: VQE can incorporate classical and quantum error mitigation strategies, improving accuracy on noisy devices.

Some Disadvantages of VQE:

Measurement Overhead: Computing expectation values requires a large number of measurements, making VQE expensive in terms of runtime.

Optimization Challenges: Classical optimization can struggle due to barren plateaus (where gradients vanish), leading to inefficient training.

Ansatz Design Trade-offs: While expressive ansatz can capture complex wavefunctions, they often require deep circuits, which are difficult to implement on NISQ hardware.

Adiabatic quantum computing

Continuous-Time Evolution in Quantum Systems We have seen that in quantum computing, the evolution of a closed system is described by a unitary transformation. That is, the state of the system $|\psi(t)\rangle$ at time t can be expressed as

$$|\psi(t)\rangle = U(t, t_0) |\psi(t_0)\rangle ,$$

where $U(t, t_0)$ is a unitary operator that depends on times t and t_0.

An equivalent way of stating this fact is the Schrödinger equation. The Schrödinger equation provides a continuous-time description of the system while through unitary operators, we have a discrete-time description:

$$H |\psi(t)\rangle = i\hbar \frac{d}{dt} |\psi(t)\rangle ,$$

where $\hbar$ is Planck's constant. In practice, it is common to absorb $\hbar$ into H, the Hamiltonian describing the system.

The Schrödinger equation is a differential equation, whose solution is given by

$$|\psi(t)\rangle = e^{-iH\cdot(t-t_0)} |\psi(t_0)\rangle .$$

One can prove that $e^{-iH(t-t_0)}$ is always unitary and any unitary can always be expressed as e^{-iK} for some Hermitian operator K. This proves the equivalence between the two alternative ways of viewing time evolution in quantum systems.

In other words, if the state of a system at time t_0 is $|\psi(t_0)\rangle$, its state at time t is given by $e^{-iH(t-t_0)} |\psi(t_0)\rangle$ and the corresponding unitary transformation is

$$U(t, t_0) = e^{-iH\cdot(t-t_0)}.$$

Note that if $|\psi\rangle$ is an eigenstate of H, it only acquires a phase when it is evolved by H, as it is not possible to transition between eigenstates.

Matrix exponentials

For a given matrix A, the matrix exponential $\exp(A)$ is defined as

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!}.$$

For instance, for

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},$$

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!} = I + A + \frac{A^2}{2!} + \cdots = I + A(e-1) = \begin{pmatrix} e & 0 \\ 0 & 1 \end{pmatrix}.$$

Hermitian matrices are diagonalizable, i.e., if A is Hermitian, A can be written as $A = PDP^{-1}$ where P is a unitary matrix. Using this fact, we can show that

$$\exp(A) = \sum_{k=0}^{\infty} \frac{(PDP^{-1})^k}{k!} = P\left(\sum_{k=0}^{\infty} \frac{D^k}{k!}\right) P^{-1} = P\exp(D)P^{-1}.$$

For a diagonal matrix

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix},$$

one can show that

$$\exp(D) = \begin{pmatrix} e^{d_1} & 0 & \cdots & 0 \\ 0 & e^{d_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{d_n} \end{pmatrix}.$$

Note that both D and $\exp(D)$ are also unitary.

Let us go through an example and compute $\exp(-iHt)$ for

$$H = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}.$$

The eigenvalues are $\lambda_1 = 4$ and $\lambda_2 = 0$, and the corresponding eigenvectors are $v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $v_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, respectively.

Thus,

$$P = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix}.$$

Now,

$$\exp(-iHt) = P\begin{pmatrix} e^{-i4t} & 0 \\ 0 & 1 \end{pmatrix} P^{-1} = \frac{1}{2}\begin{pmatrix} e^{-i4t}+1 & e^{-i4t}-1 \\ e^{-i4t}-1 & e^{-i4t}+1 \end{pmatrix}.$$

Matrix Exponentials for Pauli Operators

Let us check $\exp(-iPt)$ for Pauli operators $\sigma_x, \sigma_y, \sigma_z$. Note that $P^2 = I$ for each Pauli operator P. Therefore:

$$\exp(iAx) = \cos(x)I + i\sin(x)A,$$

for real x and matrix A such that $A^2 = I$.

Using this identity:

$$\exp(-iPt) = \cos(t)I - i\sin(t)P.$$

Now, for each Pauli matrix:

$$\sigma_x: \quad \exp(-i\sigma_x t) = \begin{pmatrix} \cos(t) & -i\sin(t) \\ -i\sin(t) & \cos(t) \end{pmatrix} = R_x(2t).$$

$$\sigma_y: \quad \exp(-i\sigma_y t) = \begin{pmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{pmatrix} = R_y(2t).$$

$$\sigma_z: \quad \exp(-i\sigma_z t) = \begin{pmatrix} e^{-it} & 0 \\ 0 & e^{it} \end{pmatrix} = R_z(2t).$$

Thus, the matrix exponentials of the Pauli matrices give the rotation matrices around the respective axes x, y, and z.

Time-Dependent Hamiltonian

A Hamiltonian may change in time and vary according to some parameters. In this case, the system evolves according to Schrödinger's equation with a time-dependent Hamiltonian:

$$H(t)\left|\psi(t)\right\rangle = i\hbar\frac{d}{dt}\left|\psi(t)\right\rangle.$$

The corresponding unitary transformation is given by

$$U(t, t_0) = \exp\left(-\frac{i}{\hbar}\int_{t_0}^{t} dt' H(t')\right).$$

Quantum Adiabatic Theorem

Let time-dependent Hamiltonian $\tilde{H}(s)$ be a Hermitian operator that varies smoothly as a function of $s := t/T$ for $s \in [0, 1]$. For T arbitrarily large, $H(t) = \tilde{H}(t/T)$ varies arbitrarily slowly.

If the system is initialized at $t = 0$ in the state $|\psi(0)\rangle$, which is the ground state of $\tilde{H}(0) = H(0)$, and $H(s)$ has a unique ground state for all s, then in the limit $T \to \infty$, the final state $|\psi(T)\rangle$ will be the ground state of $\tilde{H}(1) = H(T)$.

In simpler terms:

$$\tilde{H}(s) = (1-s)H_i + sH_f, \quad s \in [0, 1],$$

or equivalently

$$H(t) = \left(1 - \frac{t}{T}\right)H_i + \frac{t}{T}H_f, \quad t \in [0, T].$$

If the system starts in the ground state of H_i and evolves slowly enough, it ends up in the ground state of H_f.

The runtime T depends on the energy gap between the ground and the first excited state. As the gap becomes smaller, T must become larger.

Adiabatic Quantum Computing and Quantum Annealing

The quantum adiabatic theorem provides an alternative model for quantum computation. If we know the ground state of H_i and want to find that of H_f, initializing in $|\psi(0)\rangle$ (ground state of H_i) and evolving slowly ensures we end up in the ground state of

H_f.

This model of computation is called *Adiabatic Quantum Computing* (AQC) and is equivalent to the standard gate-based model.

Quantum annealing (QA) is a heuristic optimization algorithm that runs in the AQC framework. D-Wave provides quantum annealers implementing QA.

QA is a special case of AQC. In QA, the Hamiltonians are required to be *stoquastic*, i.e., $\langle i| H |j\rangle \leq 0, \forall i \neq j$, meaning the off-diagonal elements are real and nonpositive, allowing the ground state to be interpreted as a classical probability distribution.

QAOA basis

Quantum approximate optimization algorithm (QAOA) is a hybrid algorithm for solving combinatorial optimization problems, first introduced by Farhi et al. In this notebook, we will learn about its basics and motivation.

In combinatorial optimization problems, we look for the best solution among a finite set of solutions. The solution space is often exponentially large. The solution space being finite and discrete makes it harder to find the best solution. For instance, although there are efficient methods for continuous optimization problems which have infinite solution space, many of the combinatorial optimization problems belong to the class NP-hard, i.e., no efficient algorithm is known for solving them.

How QAOA Works? In QAOA, the goal is to find the ground state of H_C, which is called the problem or cost Hamiltonian. Note that QAOA is an approximate algorithm and it does not guarantee the optimal solution.

Consider a function f defined on n bits. Suppose that the Hamiltonian H_C satisfies

$$H_C|x\rangle = f(x)|x\rangle,$$

for $x \in \{0,1\}^n$. The ground state of H_C corresponds to x that minimizes $f(x)$ and the ground state energy is the minimum value of $f(x)$.

Therefore, minimizing the expectation value of H_C is equivalent to minimizing $f(x)$.

We will look at how to express a combinatorial optimization problem as a function of bit strings $f(x)$ and how to obtain the corresponding Hamiltonian H_C in the next notebook. Let us note that such Hamiltonians are always diagonal.

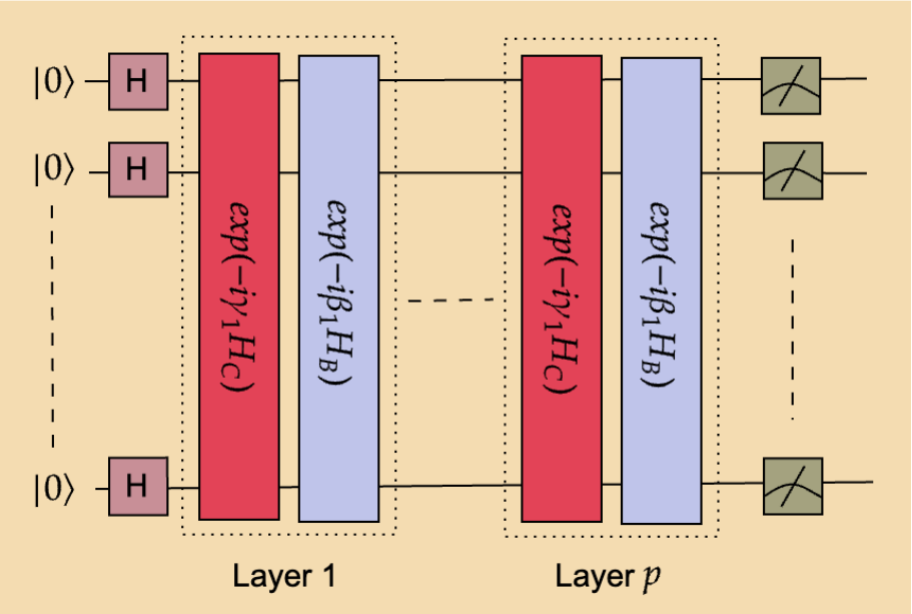
Ansatz In QAOA, the following quantum state serves as our ansatz:

$$|\psi(\gamma,\beta)\rangle = U(H_B,\beta_p)U(H_C,\gamma_p)\cdots U(H_B,\beta_1)U(H_C,\gamma_1)|s\rangle,$$

where

$$U(H_B,\beta_p) = e^{-i\beta_p H_B}, \quad U(H_C,\gamma_p) = e^{-i\gamma_p H_C},$$

and p is called the number of steps or layers that determines how many times the unitaries $U(H_B,\beta_i)U(H_C,\gamma_i)$ are applied. H_B is the mixer Hamiltonian defined as $-\sum_{i=1}^{n} X_i$ and $|s\rangle = |+\rangle^n$ is the initial state which is the ground state of H_B. β and γ form a set of $2p$ parameters.



Overall, we can express the QAOA ansatz as

$$\prod_{i=1}^{p} e^{-i\beta_i H_B} e^{-i\gamma_i H_C} |+\rangle^n.$$

In QAOA, we start with a set of initial parameters, and then iteratively measure the quantum state, and update the parameters with the help of a classical optimizer so that our cost function, which is equal to the expectation value of H_C in this case, is minimized. Note that the parameters β_p and γ_p determine the duration we evolve the state by the Hamiltonians H_B and H_C.

In QAOA as introduced by Farhi et al., $H_B = -\sum_{i=1}^{n} X_i$ and the initial state is picked as its ground state $|s\rangle = |+\rangle^n$.

In general, one can pick different H_B as long as it does not commute with H_C and pick the initial state accordingly as the ground state of H_B. This variant is called Quantum Alternating Operator Ansatz.

To understand how and why QAOA works, we need to understand some concepts.

Trotterization We can express $U(T, 0)$ as

$$U(T,0) = U(t_n, t_{n-1})U(t_{n-1}, t_{n-2}) \cdots U(t_1, t_0) = \prod_{k=1}^{n} U(t_k, t_{k-1}),$$

where $t_k = k\delta t$ $(k = 0, 1, \ldots, n)$, $t_n = T$, $t_0 = 0$, and $\delta t = T/n$. Here, we take small time steps δt to discretize the continuous evolution. Note that this step is exact for a time-independent Hamiltonian.

In our case where the Hamiltonian is time-dependent, during the small time step, we assume that the Hamiltonian is approximately constant, and approximate

$$U(t_k, t_{k-1}) \approx e^{-iH(t_k)\cdot\delta t},$$

where $H(t_k)$ is the Hamiltonian at time t_k. Hence,

$$U(T,0) \approx \prod_{k=1}^{n} e^{-iH(k\delta t)\cdot\delta t}.$$

In the case of AQC, recall that the Hamiltonian at time t is given by

$$H(t) = \left(1 - \frac{t}{T}\right) H_i + \frac{t}{T} H_f.$$

Replacing this in the above equation we get

$$U(T,0) \approx \prod_{k=1}^{n} e^{-i\left(\left(1-\frac{k\delta t}{T}\right)H_i + \frac{k\delta t}{T} H_f\right)\delta t}.$$

Now we use the Trotter formula which states that $e^{x(A+B)} = e^{xA}e^{xB} + O(x^2)$ for non-commuting Hamiltonians A and B.

$$U(T,0) \approx \prod_{k=1}^{n} e^{-i(1-\frac{k\delta t}{T})\delta t H_i} e^{-i\frac{k\delta t}{T}\delta t H_f}.$$

This equation suggests that by applying the Hamiltonians H_i and H_f iteratively, we can approximate adiabatic evolution as n gets larger and larger.

This is the inspiration behind QAOA, although it does not exactly mimic the equation above for $U(T,0)$.

QAOA Implementation

How to implement QAOA? Implementation of QAOA consists of three parts:
- Initial state - Mixer Hamiltonian - Cost Hamiltonian

Initial state
Since the initial state is $|+\rangle^n$, we can implement it by applying the Hadamard gate H to all qubits.

Mixer Hamiltonian
The mixer Hamiltonian is defined as

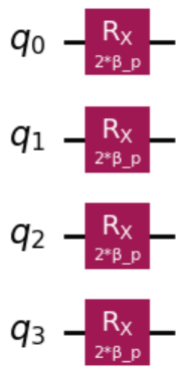
$$H_B = -\sum_{i=1}^{n} X_i$$

and the corresponding unitary at layer p is given by

$$U(H_B, \beta_p) = e^{-i\beta_p H_B}.$$

The Hamiltonian $H_B = -\sum_{i=1}^{n} X_i$ consists of Pauli X-operators acting on each qubit. Using the identities from the previous notebook, we can express it as

$$U(H_B, \beta_p) = e^{-i\beta_p H_B} = \prod_{i=1}^{n} R_{x_i}(2 \cdot \beta_p).$$



Cost Hamiltonian

In QAOA, the cost Hamiltonian is diagonal and consists of Z terms only. Here is an example cost Hamiltonian:

$$H_C = Z_0Z_1 + 3Z_1Z_2 + 2Z_0Z_2.$$

It is common to have Hamiltonians that consist of terms that act on at most two qubits, i.e., 2-local (the reason will be explained in the following notebook), but in QAOA one can also implement multi-qubit interactions. For simplicity, let us assume that our Hamiltonian is 2-local. In that case,

$$H_C = \sum_i h_i Z_i + \sum_{i,j} J_{ij} Z_i Z_j.$$

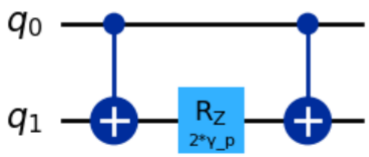
Note that single-qubit Z terms and 2-qubit terms commute. Therefore, we can use the identity $e^{A+B} = e^A e^B$. Hence, the corresponding unitary is given by

$$U(H_C, \gamma_p) = e^{-i\gamma_p H_C} = e^{-i\gamma_p \sum_i h_i Z_i} e^{-i\gamma_p \sum_{i,j} J_{ij} Z_i Z_j} = \prod_i R_{Z_i}(2 \cdot h_i \gamma_p) \prod_{i,j} R_{Z_i Z_j}(2 \cdot J_{ij} \gamma_p).$$

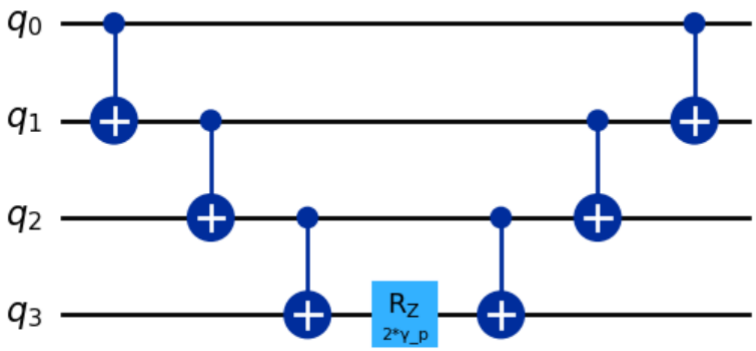
The two-qubit rotation $R_{ZZ}(\theta)$ is defined as

$$R_{ZZ}(\theta) := e^{-i\frac{\theta}{2} Z \otimes Z} = \begin{pmatrix} e^{-i\theta/2} & 0 & 0 & 0 \\ 0 & e^{i\theta/2} & 0 & 0 \\ 0 & 0 & e^{i\theta/2} & 0 \\ 0 & 0 & 0 & e^{-i\theta/2} \end{pmatrix}.$$

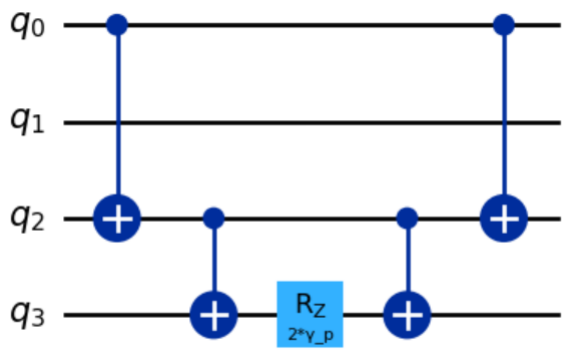
In Qiskit, it can be directly implemented by the ‘rzz‘ gate.



The construction can be generalized to multiple qubits. For example: - Circuit for the term $Z_0Z_1Z_2$



- Circuit for the term $Z_0Z_2Z_3$



Cost function

Let us recall how the cost Hamiltonian is defined:

$$H_C|x\rangle = f(x)|x\rangle.$$

The cost function $f(x)$ depends on the problem we want to optimize.

Circuit measurement yields a bit string x' which encodes a solution to the problem. Using this bit string, one can calculate the corresponding cost $f(x')$. This is needed to estimate the expectation value of H_C from the measurement results.

Example — Max-Cut problem

Given a graph, the problem requires splitting the vertices/nodes into two disjoint groups so that there are as many edges as possible between the groups. The partition of two adjacent vertices into disjoint sets is called a cut. The goal is to find a cut that contains the maximum number of edges.

Let us encode the solution space of this problem using bit strings. Suppose that for a bit string $x_1x_2\cdots x_n$, - Node i belongs to group 0 if $x_i = 0$, - Node i belongs to group 1 if $x_i = 1$.

Given a bit string, we should be able to count the number of edges in the cut and determine the associated cost. An edge belongs to the cut if its endpoints belong to different groups. For two vertices i and j,

$$(i,j) \text{ belongs to the cut if } x_i + x_j - 2x_ix_j = 1.$$

So a summation over all edges E in the graph gives us the number of edges in the cut:

$$\sum_{(i,j)\in E} (x_i + x_j - 2x_ix_j).$$

Since in QAOA we are trying to minimize the cost function, we put a minus sign to get the cost function:

$$f(x) = -\left(\sum_{(i,j)\in E} (x_i + x_j - 2x_ix_j)\right).$$

This is how we can define our cost function for the Max-Cut problem.

The algorithm

Let us put the pieces together now. We denote the parameters at iteration i by (γ_i, β_i).

We construct our ansatz

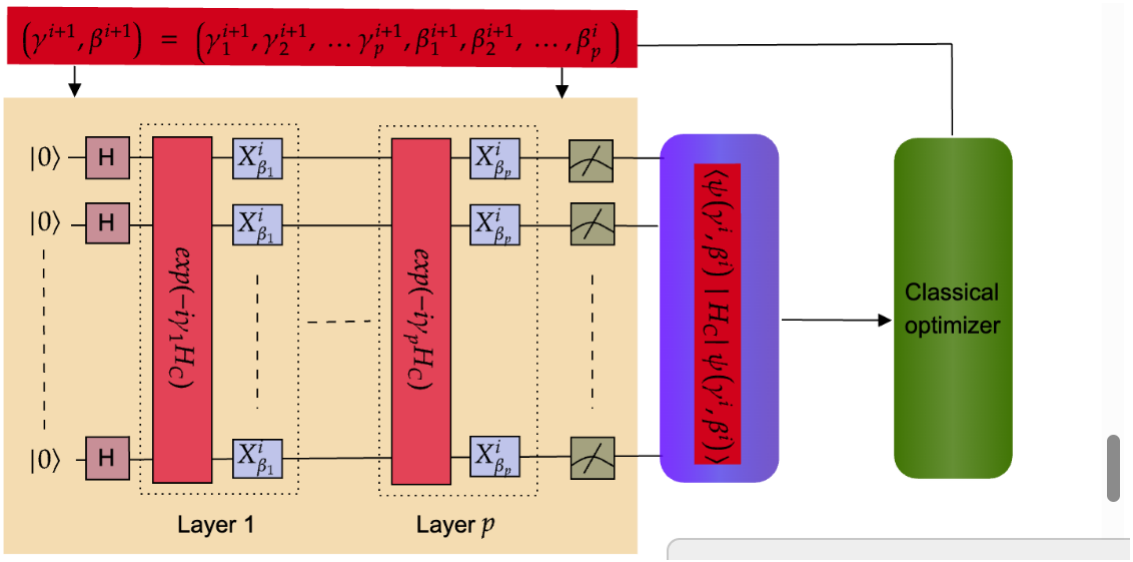
$$\prod_{p=1}^{P} e^{-i\beta_p H_B} e^{-i\gamma_p H_C} |+\rangle^n.$$

We initialize its parameters to obtain $|\psi(\gamma_0, \beta_0)\rangle$.

We use a classical optimizer to adjust the parameters γ and β to minimize our cost function. The cost function is computed by the outcomes of the measurement results, and its minimization corresponds to the minimization of

$$\langle\psi(\gamma_i, \beta_i)|H_C|\psi(\gamma_i, \beta_i)\rangle.$$

QAOA is a hybrid algorithm, where the optimization of the parameters is performed on a classical computer, and the quantum computer is used for computing the cost function.



Advantages of QAOA for NISQ Devices: - **Near-Term Applicability:** QAOA is designed for near-term quantum devices as it doesn't require full fault tolerance and can be run with relatively shallow circuits. - **Problem-Specific Ansatz:** QAOA can be tailored to specific optimization problems by adjusting the structure of the quantum circuits. - **Hybrid Approach:** Like VQE, QAOA is a hybrid quantum-classical algorithm.

Disadvantages of QAOA: - **Measurement Overhead:** QAOA requires many measurements to estimate expectation values, leading to high computational cost. - **Optimization Challenges:** The classical optimization can be difficult due to barren plateaus and complex cost landscapes. - **Ansatz Depth Trade-offs:** Deeper circuits improve performance but are more susceptible to noise and gate errors on NISQ devices.

Ising Model

Electrons have a quantum mechanical property called **spin**, which is the angular momentum of the particle. When measured, it is either $\frac{h}{2}$ (spin up) or $-\frac{h}{2}$ (spin down), where h is Planck's constant.

An electron's spin is closely related to its magnetic moment so that an electron behaves like a tiny bar magnet with a North (N) and a South (S) pole. Ferromagnetism arises when a collection of atomic spins align such that their associated magnetic moments all point in the same direction, and the spins behave like a big magnet with a net macroscopic magnetic moment.

Image from *General Chemistry, 3rd edition, by Hill and Petrucci.*

The Ising Model is a mathematical model to study ferromagnetism in statistical physics. The Ising model was first proposed by Wilhelm Lenz, who gave it as a problem to

his graduate student Ernst Ising, after whom this model is named. You can check this link for more information.

For simplicity, we will say each spin takes either the value $s = 1$ (up) or $s = -1$ (down).

When spins are arranged on a 1-D line, so that each spin interacts only with its right and left neighbors, the model is called the **1-Dimensional Ising Model**.

When the spins are arranged on a 2-D lattice, each spin interacts with its right, left, up, and down neighbors, the model is also known as the **2-Dimensional Ising model**.

The configuration of spins yielding the lowest energy is known as the **ground state**. It is NP-Hard to find the ground state of a 2-D Ising model. Thus, finding the ground state is as hard as problems like the Max-Cut problem and the Travelling Salesperson problem.

Note that spins can be arranged in any other configuration.

The energy of the system

We would like to express the energy of every possible configuration of the spins in the system. We will assume that all possible couplings are possible between any two spins (for example, J_{ij} may exist for any (i, j) pair). In realistic scenarios, it may not be possible to have a coupling between any two pair of qubits.

- The spins interact with the external magnetic field h, if present.
- Each spin state (variable) interacts with the other spins. The coupling strength of this spin-spin interaction is characterized by the constant J.
- Each spin variable s_i takes the values $\{-1, 1\}$.

Based on those assumptions, the energy of the Ising Model is given by

$$E_{\text{Ising}}(s) = \sum_i h_i s_i + \sum_{i,j} J_{ij} s_i s_j$$

The energy of the cost Hamiltonian

Recall that the 2-local cost Hamiltonian is expressed as follows:

$$H_C = \sum_i h_i Z_i + \sum_{i,j} J_{ij} Z_i Z_j$$

Note that the eigenvectors of state Z are $|0\rangle$ and $|1\rangle$, with eigenvalues 1 and -1, respectively. Therefore, when H_C is applied on some state $|\psi\rangle$, we get the following state:

$$H_C |\psi\rangle = \sum_i h_i Z_i |\psi_i\rangle + \sum_{i,j} J_{ij} Z_i |\psi_i\rangle Z_j |\psi_j\rangle$$

$$= \sum_i h_i (-1)^{\psi_i} |\psi_i\rangle + \sum_{i,j} J_{ij} (-1)^{\psi_i} (-1)^{\psi_j} |\psi_i\rangle |\psi_j\rangle$$

$$= \left(\sum_i h_i s_i + \sum_{i,j} J_{ij} s_i s_j \right) |\psi\rangle$$

where $|\psi_i\rangle$ is the state of qubit i and $s_i \in \{-1, 1\}$ are spin variables.

Note that the energy of the state is exactly the energy of the Ising model we have defined previously:

$$E_{\text{Ising}}(s) = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j$$

Hence, the minimization of $E_{\text{Ising}}(s)$ is equivalent to the minimization of the energy of the cost Hamiltonian. This suggests that by expressing our problems in the form of Ising models, we can easily get a corresponding cost Hamiltonian.

The Ising formulation for the Max-Cut Problem

The Max-Cut problem results in a natural Ising model formulation. Let us recall the definition.

Given a graph, the problem requires splitting the vertices/nodes into two disjoint groups so that there are as many edges as possible between the groups. The partition of two adjacent vertices into disjoint sets is called a cut. The goal of this problem is to find a cut in such a way that the cut covers the maximum number of edges.

For each vertex i, we will use a spin variable s_i to decide which group it should belong to:

$$s_i = \begin{cases} 1, & \text{if vertex } i \text{ is in Group 0} \\ -1, & \text{if vertex } i \text{ is in Group 1} \end{cases}$$

Our objective is to maximize the number of edges in the cut.

Let E be the set of edges. Note that for an edge (i, j), $s_i s_j = 1$ if the vertices are in the same group and $s_i s_j = -1$ otherwise.

Hence, we can express the exact number of edges in the cut as

$$\frac{1}{2} \sum_{(i,j)\in E} (1 - s_i s_j)$$

which is a maximization problem. The equivalent minimization problem is given by

$$\text{mín}\, \frac{1}{2} \sum_{(i,j)\in E} (s_i s_j - 1)$$

Note that in practice, it would be enough to minimize

$$\text{mín} \sum_{(i,j)\in E} s_i s_j$$

since -1 is just a constant term. So, the spin configuration minimizing the energy of the above problem yields the optimal solution to the Max-Cut problem.

Hence, we can conclude that the cost Hamiltonian for the Max-Cut problem is defined as

$$H_C = \sum_{(i,j)\in E} Z_i Z_j$$

M8: Quantum Machine Learning

textbfMachine Learning (ML) is a subset of artificial intelligence (AI) that focuses on developing systems that can learn from data and improve their performance over time. By using algorithms to identify patterns within data, ML models can make predictions or decisions without being explicitly programmed. This ability to learn and adapt makes ML a powerful tool across various industries, including finance, healthcare, and technology.

Types of Machine Learning

There are several types of machine learning, each with different methodologies and applications.

- **Supervised Learning:** This type involves training a model on a labelled dataset, where the desired output is known. The model learns to map inputs to outputs based on the example pairs. Common algorithms include linear regression, decision trees, and support vector machines. Supervised learning is often used in applications such as email spam detection, fraud detection, and predictive maintenance.
- **Unsupervised Learning:** The model identifies patterns and relationships within an unlabelled dataset, often used for clustering or association tasks. Algorithms like k-means clustering, hierarchical clustering, and principal component analysis (PCA) are commonly used. Unsupervised learning is useful in customer segmentation, market basket analysis, and anomaly detection.
- **Reinforcement Learning:** A learning paradigm where an agent learns to make decisions by performing actions and receiving feedback through rewards or penalties. This type of learning is inspired by behavioural psychology and is often used in robotics, game playing, and autonomous vehicles. Popular algorithms include Q-learning and deep Q-networks.

The Machine Learning Process

The ML process typically involves several stages, each essential to building an effective model.

1. **Data acquisition:** Collecting relevant data from various sources, such as databases, CSV files, sensor feeds, or web scrapping.
2. **Data pre-processing:** Cleaning, transforming, and preparing data from the acquired data. This can involve removing outliers, inputting missing data, and encoding categorical variables.
3. **Feature selection:** In this stage, the most relevant features for the problem at hand are selected. This may involve techniques such as PCA, feature selection based on importance, or removal of highly correlated features. This process is known as embedding and the idea is to spread or rearrange data to show hidden trends or information.
4. **Model selection:** Once the data is prepared, the appropriate machine learning model for the problem at hand is selected. This can range from simple algorithms like linear regression or decision trees to more complex models like neural networks or deep learning algorithms.

5. **Model training:** With the selected model and data. The training adjusts its parameters to minimise the difference between the training dataset and a real dataset.

6. **Model evaluation:** Once trained, the model's quality is tested based on accuracy, recall, F1-score, and ROC curves.

7. **Model tuning:** If the model's performance isn't strong enough, adjustments are made iteratively until the quality is acceptable.

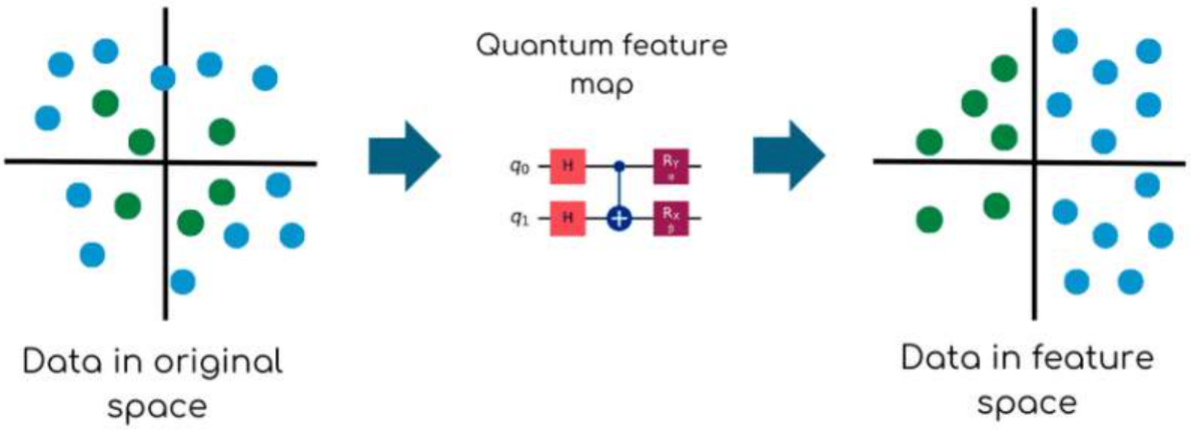
8. **Model deployment:** And finally, once the model is acceptable, it can be deployed into production. Models should be routinely monitored and maintained to ensure its continuous performance.

Improving Machine Learning with Quantum Computing

QML is not an entirely quantum process. Instead, it is a hybrid approach that leverages the strengths of quantum computing to enhance traditional machine learning models. The integration of quantum computing is typically limited to specific stages of the machine learning process where it can provide significant improvements. In general, the blocks of the ML process that can be modified to incorporate quantum enhancements include feature selection and model selection.

Step 3. Feature Selection

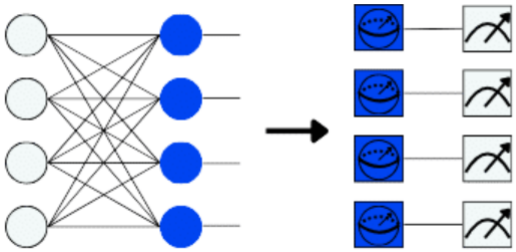
To encode data and adapt it to the architecture presented by quantum computers, it's necessary to perform different data processing than usual, leveraging both the improvements and limitations provided by quantum computing. This feature extraction is usually condensed into what is known as a feature map, which is a quantum circuit that encodes data appropriately to benefit from the larger space provided by qubits. With a higher-dimensional space available, data can be distributed in various ways to try to find common features among them or hidden trends not immediately visible.



Step 4. Model Selection

To transform into a QML process, it is necessary to remove the classical model and

introduce a quantum one. The transition from a classical computational model to a quantum one entails replacing the existing framework with a generalised quantum circuit. This new model incorporates quantum gates that are parameterised by angles, providing the degrees of freedom necessary for the model to explore potential solutions to a given problem. To effectively utilise quantum computing capabilities, a feature map is employed to transform classical data into quantum states. This transformation involves spreading or rearranging the data to optimise its compatibility with quantum algorithms. In the quest for the optimal solution, the model iteratively adjusts the angles of the quantum gates within the circuit. However, the effectiveness of QML models depends heavily on the specific problem and the quantum circuit's design and capabilities.



Advantages of Quantum ML

Quantum Machine Learning combines the strengths of both quantum computing and classical machine learning, resulting in several advantages.

Improved Model Accuracy
Faster Computation Times
Innovation in Algorithm Development

Applications of QML

Data Science and Business Analytics

Machine learning has revolutionised data science and business analytics by enabling sophisticated data analysis, predictive modelling, and pattern recognition. Quantum machine learning can further enhance these capabilities by processing vast datasets more efficiently and uncovering complex patterns that classical algorithms may miss. The increased computational power of quantum systems allows for faster data processing and more accurate predictions, leading to better-informed business decisions and strategies.

Artificial Intelligence

In artificial intelligence, machine learning algorithms have paved the way for advancements in natural language processing, computer vision, and decision-making processes with LLMs. Quantum machine learning can take AI to new heights by speeding up the training of models and optimising neural networks beyond classical limits. This can result in more intelligent and responsive AI systems capable of solving previously insurmountable problems and providing deeper insights into human-like cognition.

Finance

The finance sector benefits from machine learning through algorithmic trading, risk management, fraud detection, portfolio optimisation, and credit scoring. Quantum machine learning can significantly improve financial models by handling large-scale simulations

and computations more efficiently, offering better risk assessments and quicker transaction processing. This could lead to more robust financial systems capable of responding to market changes with greater agility and accuracy.

Automation

Automation has seen significant advancements due to machine learning's ability to learn from data and make real-time decisions. Quantum algorithms can solve intricate optimisation problems faster and more accurately, ensuring more efficient operations and reduced costs in automated systems by optimising complex logistics, supply chain management, and manufacturing processes.

Telecommunication

The telecommunications industry relies heavily on data processing and efficient communication networks. QML can impact telecom by enhancing network optimisation and improving signal processing. With the ability to process and analyse extensive datasets swiftly, QML can optimise bandwidth allocation, reduce latency, and improve overall service quality. Furthermore, QML can enhance cybersecurity measures within telecom networks by identifying and mitigating threats more effectively, ensuring more secure and reliable communication infrastructures.

Healthcare

In healthcare, machine learning is already being used for diagnostics, personalised medicine, and drug discovery. QML has the potential to transform healthcare by offering faster and more accurate diagnostic tools, optimising treatment plans, and accelerating the drug development process. QML could analyse complex medical data sets, such as genomic data and medical imaging, to uncover patterns and insights that classical algorithms might miss. This can lead to early disease detection, tailored therapies, and improved patient outcomes.

Introduction to Optimisation Algorithms in QML

Optimisation techniques are essential because they allow quantum machine learning models to be fine-tuned, improving their accuracy and efficiency. We will see how these algorithms can be applied to optimise various parameters in quantum circuits, enabling better performance for tasks such as classification, regression, and clustering. Understanding these optimisation methods is a key skill for anyone involved in QML, as they underpin the ability to solve complex problems efficiently.

Parametric Quantum Circuits

The conversion of QUBO problems to Ising models enables quantum hardware to tackle complex combinatorial optimisation challenges efficiently, as demonstrated by the mathematical transformation that maps binary variables to spin states. This foundational technique, critical for implementing quantum annealing and variational algorithms, expands QML applications to domains like logistics optimisation and quantum neural network training.

Parametric quantum circuits (PQCs) enhance this framework by integrating tunable parameters that optimise quantum states during ML training. Recent advancements like Quantum Parameter Adaptation (QPA) leverage PQCs to generate classical model weights during training while decoupling inference from quantum hardware, achieving 80–90 % parameter reduction in large language models. Innovations such as batched parameter generation further enable near-term quantum compatibility by reducing qubit requirements from logarithmic to constant scales. Hybrid architectures like SPQCC com-

bine per-channel PQCs with classical post-processing, achieving state-of-the-art accuracy on benchmarks like MNIST while maintaining scalability. Theoretical improvements in PQC trainability now use reduced-domain parameter initialisation to avoid barren plateaus, enabling polynomial-time convergence even for deep circuits.

These developments collectively address historical QML limitations in encoding overhead and quantum dependency, while enabling fine-grained exploration of high-dimensional quantum landscapes through optimised parametric control.

These circuits typically involve gates, like Rx, Ry, and Rz, dependent on parameters such as rotation angles, which can be continuously adjusted. By optimising these parameters, we can tailor the quantum circuit for specific tasks. Different applications of parametric circuits, such as encoding data or creating neural networks, are fundamental for developing effective QML models.

Variational Quantum Eigensolver

As we have seen in Module 6, VQE is a powerful hybrid quantum-classical algorithm designed to estimate the ground state energy of quantum systems. Its versatility makes it valuable in quantum chemistry, for which it was originally developed to find the ground state of He–H^+, materials science, and even QML. In VQE, we start with an *ansatz* (a parameterised quantum circuit that represents a guess for the system's quantum state). We then adjust the circuit's parameters (like tuning knobs) to find the lowest possible energy. This is done by repeatedly measuring the circuit's output and updating the parameters to minimise the energy—an approach that extends naturally to solving QUBO and Ising models. At its core, VQE relies on the variational principle, which guarantees that the energy expectation value of any trial wavefunction serves as an upper bound to the true ground state energy. Through iterative parameter optimisation, the algorithm refines this wavefunction to approximate the lowest-energy solution.

To use the VQE algorithm for QML, follow these steps:

1. **Initialise parameters:** Initial values for the variational parameters are set classically. These initial values implement the form of the quantum state to be generated, determined by the chosen ansatz.

2. **Introduce the parameters into the circuit:** The determined parameters are initialised into the quantum circuit through variational gates. These parameters specify how the qubits should be manipulated.

3. **Expectation value:** The expectation value of the circuit is then calculated based on the energy of the final state. This is done through measuring the circuit and computing it classically.

4. **Update the parameters:** Using the expectation value, classical gradient optimisers adjust the variational parameters to further minimise the system's energy.

5. **Repeat from step 2:** Iterate from step 2 to step 4 multiple times until the system converges to a minimum energy. This minimum energy is a good approximation for the optimal solution of the problem.

Optimisers

To be able to implement step 4 above, gradient descent optimisers play a crucial role in updating the parameters. Two notable optimisation techniques often used in QML are

COBYLA (Constrained Optimisation BY Linear Approximations) and SPSA (Simultaneous Perturbation Stochastic Approximation).

COBYLA

COBYLA is a numerical optimisation method that does not require gradient information but instead relies on linear approximations to constrain the search space. It is particularly useful for problems with complex constraints and can efficiently manage high-dimensional parameter spaces. For QML, COBYLA is employed to optimise the variational parameters of quantum circuits, allowing for the exploration of the energy landscape without the need for explicit gradient computations. This makes it suitable for NISQ devices, where obtaining accurate gradients can be challenging due to measurement errors and noise.

SPSA

SPSA, on the other hand, is a stochastic optimisation technique that estimates the gradient of the objective function using random perturbations. Similar to COBYLA, this method is highly efficient in high-dimensional settings, as it requires only a few function evaluations to approximate the gradient, making it computationally less expensive than traditional gradient descent methods. SPSA is especially useful in quantum machine learning because it reduces the number of times you need to measure costly quantum states (like the energy of a system). This saves time and resources. SPSA uses small, random adjustments to explore the energy landscape of a quantum problem. This helps it avoid getting stuck in local optima, especially when noise or rough terrain (like jagged energy curves) make optimisation tricky.

Quantum Approximate Optimisation Algorithm

QAOA is a powerful hybrid quantum-classical algorithm originally developed for combinatorial optimisation problems. However, its versatility extends beyond optimisation, making it a valuable tool in QML. Let us explore how QAOA can be applied to QML tasks, enhancing performance and enabling new computational paradigms.

Applications of QAOA

- Feature Selection
- Clustering and Community Detection
- Training Quantum Neural Networks
- Adversarial Learning and Robust Models

QAOA in Financial Portfolio Optimisation

One of the most notable real-world applications of the QAOA is in financial portfolio optimisation. Financial institutions constantly seek efficient methods to allocate assets in a portfolio to maximise returns while minimising risks and variation. The ratio between these two variables is known as the Sharpe ratio, which is the metric financial firms look at when evaluating portfolios. Traditional optimisation techniques often face challenges due to the vast number of possible asset combinations and the complexity of market dynamics.

In this context, QAOA has been employed to tackle the combinatorial nature of portfolio optimisation, where choosing which stocks, bonds, or materials to add and how much of them can impact the rate of return significantly. By leveraging QAOA, financial analysts

can explore a multitude of potential asset allocations simultaneously, significantly reducing the time required to identify optimal solutions. While companies such as DATEV and BBVA are partnering with quantum computing companies to implement this problem, it is still being actively researched at many different universities, such as the University of Salamanca.

Consider a financial firm aiming to optimise a portfolio comprising 50 different assets, where each asset must be allocated at least 1 % and weights are in whole percentages. Using classical methods, this problem would require evaluating an astronomically large number of combinations, upwards of $5{,}04 \times 10^{28}$, making it computationally infeasible for larger asset pools. Financial portfolios, funds, or ETFs can have upwards of 7000 different assets under management, so 5×10^{28} combinations could be on the lower end. These ballooning numbers are one of the reasons why the financial industry was one of the first to begin developing their quantum workforce.

QAOA vs. VQE

When comparing QAOA and VQE, the Hamiltonian structure plays a pivotal role. QAOA streamlines operations by encoding the Hamiltonian into a single circuit through exponential construction, whereas VQE executes separate circuits for each Hamiltonian term or block. This difference has economic implications for users on platforms like Amazon Braket, where costs scale with the number of circuits and shots (repetitions). Reducing circuit count—as QAOA does—can lower expenses, giving it a practical edge in cost-sensitive applications.

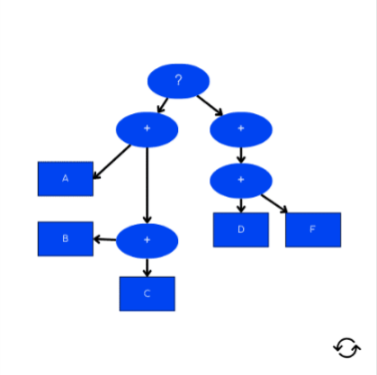
However, noise resilience is another critical factor. Quantum devices prone to errors may favour VQE due to its simpler computational demands, as shorter circuit depths reduce error accumulation. Both methods prioritise near-optimal solutions over perfection, aiming for energy levels close to the theoretical minimum. For example, in problems like Sudoku, where exactness is non-negotiable, these algorithms focus on achieving results within an acceptable margin of error rather than guaranteeing perfection.

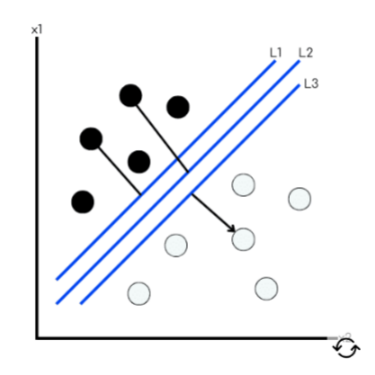
This balance of cost-efficiency and noise tolerance highlights the trade-offs in quantum optimisation. While QAOA's streamlined approach saves resources, VQE's robustness in noisy environments ensures reliability, underscoring the need to align method selection with hardware capabilities and problem constraints.

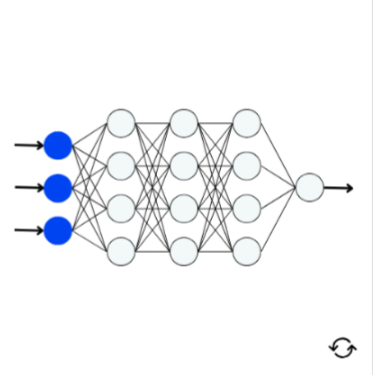
Classical Machine Learning: Classification and Regression

Classification

Classification is a supervised learning technique where the goal is to assign labels to instances based on input features. This process involves training a model on a labelled dataset, where each instance is associated with a known label. The model learns to recognise patterns and relationships within the data that correspond to each label. Once trained, the model can predict the labels for new, unseen instances. Common algorithms used for classification include:







Decision Trees: A tree-like model used to make decisions based on input features. Decision trees split the data into branches to reach a final classification decision.

Support Vector Machines (SVM): A model that finds the hyperplane which best separates different classes. SVMs are effective in high-dimensional spaces.

Neural Networks: Multi-layer models that learn to classify data through backpropagation. Neural networks can model complex relationships in data.

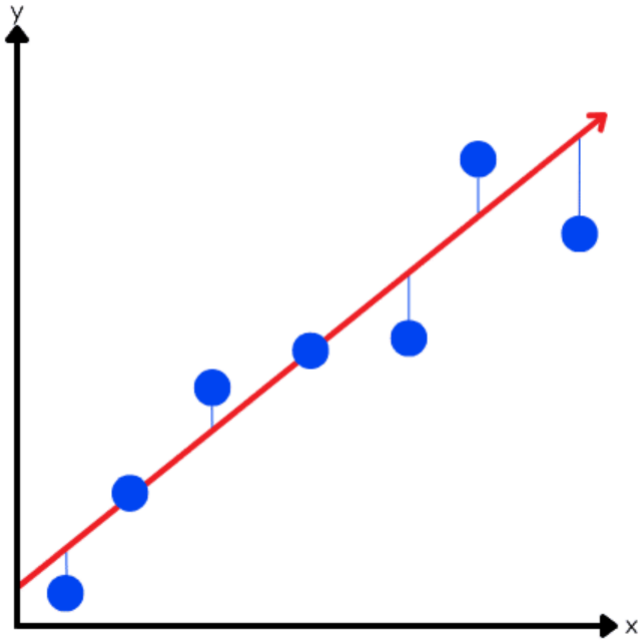
Classification models are ubiquitous in our daily lives:
Image Recognition
Spam Detection
Voice Assistant

Regression

Regression is another supervised learning technique, but here, the goal is to predict continuous values rather than discrete labels. Common algorithms include:

- **Linear Regression:** A fundamental method that fits a linear relationship between the input features and the output. It operates under the assumption that there is a straight-line relationship between the input variables and the target variable. Due to its simplicity and interpretability, linear regression is widely used in scenarios where the relationship is expected to be linear, such as house prices based on size, forecasting sales based on advertising spend, or examining the relationship between study time and exam scores.
- **Polynomial Regression:** This is an extension of linear regression that fits a polynomial relationship between the input variables and the output. By introducing polynomial terms, the model can capture non-linear patterns in the data. It involves transforming the original features into polynomial features, enabling the model to fit curved lines to the data.
- **Neural Networks:** These are a class of models inspired by the human brain, con-

sisting of multiple interconnected layers of nodes (or neurons). When applied to regression tasks, neural networks use appropriate loss functions, such as mean squared error, to predict continuous values. They are particularly powerful for capturing highly complex relationships within the data, making them suitable for tasks where traditional polynomial regression may fall short.



Applications of regression are extensive and span various fields, including finance, meteorology, and real estate. For instance:

Predicting Stock Prices

Weather Forecasting

Real Estate Valuation

We can think of classification and regression mathematically as finding a function $f(x)$, where x is a vector representing our input, such that $f(x) = y$, where y is the label associated with that vector.

- The model can be thought of as the function f that connects our inputs, x, to their labels, y.
- The dataset that is provided to the model consists of pairs of x's and y's. This is the data that is used to train the model to find parameters that best approximate the function f.
- An error function then determines the difference between the predicted output with the actual output of the dataset, $f(x) - y$. This evaluates the performance of our trained model and helps us determine what needs to be done to improve it.

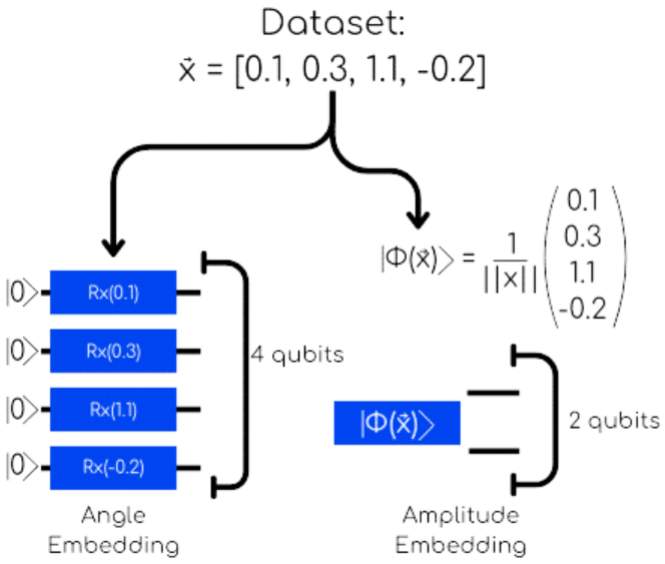
Quantum Models for Supervised Learning

In recent years, the need for models that surpass the capabilities of classical computers has become increasingly evident, especially with the rise of large language models (LLMs). QML aims to tackle problems that classical ML models struggle to handle. The choice of a quantum model depends on the specific problem at hand, making it crucial to understand each model's structure for effective selection and implementation.

An ideal quantum model should be both flexible enough to approximate the desired solution and constrained enough to maintain narrow probability distributions, ensuring efficient identification of the best candidate solution.

Quantum Embedding

Embedding is an important process of creating a QML model. Embedding is the process by which we encode real data inside quantum computers. Two such strategies are: amplitude embedding and angle embedding.

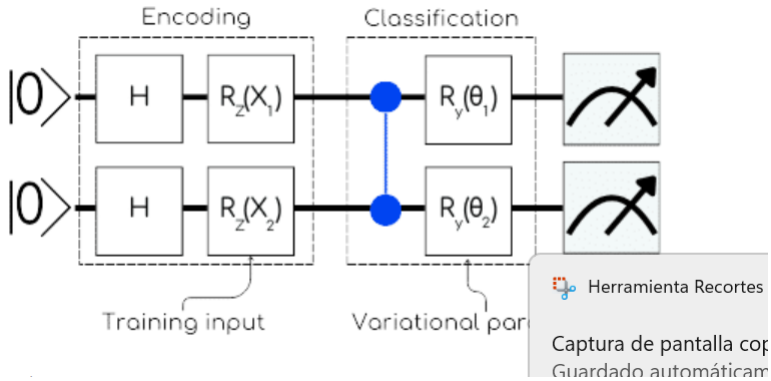


- The first strategy, amplitude embedding, involves normalising the dataset and using each data point as a qubit's probability amplitude.
- Angle embedding involves assigning each feature to a phase of a rotation gate family. Applying multiple layers of these gates to the same qubit yields a more optimal approach than amplitude encoding.

Quantum Classification

Quantum classification involves using quantum algorithms and quantum circuits to perform classification tasks. Some quantum approaches include:

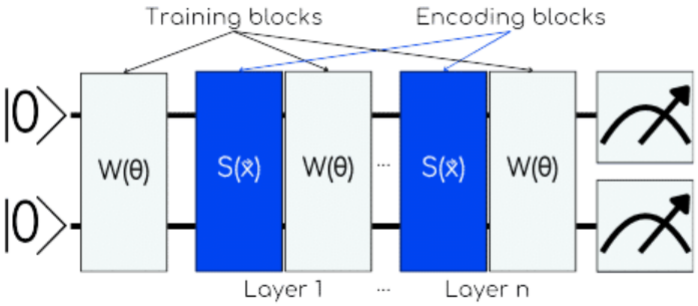
- **Quantum Support Vector Machines (QSVM):** Uses quantum kernels, a method to transform our data, and feature maps (a specific type of encoding designed to map data into a quantum Hilbert space) to classify data in high-dimensional spaces efficiently.
- **Quantum Neural Networks:** A classical neural network has series of hidden layers that get trained to produce the output we want. Instead, quantum neural networks utilise rotation gates as the various trainable nodes, which then get entangled together to create the connections between the nodes that are typically seen in classical neural networks. Recent advances have shown promise in generative AI, information processing, and more.
- **Quantum k-Nearest Neighbours (Qk-NN):** Leverages quantum distance measures (the distance between two quantum data points) to classify data points based on their nearest neighbours.



Quantum Regression

Quantum Regression Models extend classical regression techniques by leveraging quantum computing to process and analyse data more efficiently. The goal of regression, both classical and quantum, is to find relationships between input and output variables and make accurate predictions. Quantum regression can be implemented using quantum feature maps, variational quantum circuits, or hybrid quantum-classical approaches. These models can capture intricate patterns and relationships within the data that traditional models might miss, paving the way for advancements in fields such as finance, healthcare, and scientific research.

- **Quantum Linear Regression:** Utilises quantum algorithms, such as the Harrow-Hassidim-Lloyd (HHL) Algorithm, to solve linear regression problems more efficiently than classical approaches.
- **Quantum Polynomial Regression:** Employs quantum-enhanced techniques, such as variational quantum circuits, quantum least squares solvers, or quantum kernel methods, to fit polynomial functions by mapping classical data into a high-dimensional Hilbert space and optimising model parameters using quantum-classical hybrid approaches.



Quantum Training of QML Models

Now once we have a chosen quantum model and dataset, we need to train the model and test whether the model is accurate enough for our use case. This process involves two stages:

1. Training
2. Testing

Training

In the first stage, quantum training is performed through an iterative hybrid process

that integrates both classical and quantum computations. During this phase, an optimisation loop is executed to determine the optimal parameters for the quantum model. This involves adjusting the quantum circuit structure, fine-tuning variational parameters, and selecting appropriate observables to extract meaningful information. The objective is to shape the solution space in a way that allows the model to generalise well to unseen data while maintaining computational efficiency.

There are two important elements we need to train our models:

Error function

Gradient function

An error function, also known in machine learning as a cost function, quantifies how far the model's predictions deviate from the expected outputs. In quantum machine learning, common error functions include the mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks. The choice of the error function depends on the specific learning objective and directly influences the optimisation process.

A gradient function is used to update the model parameters by computing how changes in these parameters affect the error function. In classical machine learning, gradients are typically calculated using backpropagation and automatic differentiation. In quantum models, however, gradients are obtained through techniques such as parameter shift rules or finite difference methods, as quantum circuits are inherently different from classical networks. These quantum gradient estimation methods allow us to iteratively minimise the error function, improving the model's ability to make accurate predictions.

By combining these two elements—error measurement and gradient-based optimisation—the quantum training process refines the model parameters to achieve better generalisation and performance on unseen data.

Testing

In the testing stage, the trained quantum model is evaluated using new input data that it has never encountered before. This phase assesses the model's predictive accuracy and ensures that it has learned meaningful patterns rather than simply memorising training data. The primary goal is to detect and mitigate adaptive phenomena such as overfitting, verifying that the model can generalise effectively to real-world scenarios.

What is data embedding in QML?

Data embedding involves representing classical data in a form that can be processed by a quantum computer. This is a crucial step in utilising quantum computing for machine learning tasks, as it allows the integration of quantum advantages in data manipulation and analysis. Here, we will explore why data embedding is necessary and how it is achieved in quantum systems.

Quantum embedding techniques

In earlier modules, we briefly touched upon amplitude and phase encoding. Now, we will explore these and additional embedding techniques in greater detail behind each method.

Amplitude encoding

Phase encoding

Basis encoding

Angle encoding

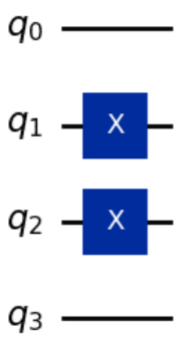
IQP style encoding

Basis encoding

Basis encoding, one of the easier encoding methods to utilise, uses the basis states of a quantum system to represent data. This method is straightforward but can become inefficient for larger datasets due to the exponential growth in the number of basis states.

For a dataset $x = \{x_1, ..., x_n\}$, the basis encoding can be represented as:$|\psi\rangle = |x_1 x_2 ... x_n\rangle$

Each data element x_i is mapped to a basis state $|x_i\rangle$. While this method is simple to implement, it requires a significant number of qubits to represent larger datasets. For a bit string of $x = 0110$, we would apply X-gates to the corresponding qubits:




```
from qiskit import QuantumCircuit

qc = QuantumCircuit(4)

qc.x(1)
qc.x(2)

qc.draw("mpl")
```

Which of the following series of gates encodes the state 1110?

```
qc.x(0)
qc.x(1)
qc.x(3)
```

```
qc.x(1)
qc.x(2)
qc.x(3)
```

```
qc.x(0)
qc.x(1)
qc.x(2)
```

Amplitude encoding

Amplitude encoding is a method where data is encoded into the amplitude of quantum states. This technique is particularly useful for efficiently representing large datasets. The mathematical representation of amplitude encoding involves normalising the data to fit within the constraints of a quantum state. Given a dataset $x = \{x_1, ..., x_n\}$, the corresponding quantum state can be written as:

$$|\psi\rangle = \sum_{i=1}^{n} x_i \, |i\rangle$$

Here, the data elements are represented, and the basis states are denoted. Each data element is mapped to the amplitude of a quantum state, and the entire dataset is normalised such that the sum of squared amplitudes equals one.

As we have seen in other modules, if a vector represents the data ($x = [0{,}5, 0{,}5, -0{,}5, -0{,}5]$), then the corresponding state would be prepared accordingly ($|x\rangle = 0{,}5\,|00\rangle + ...$). But this can't be represented as a trivial circuit; instead, we will prepare it using Qiskit's arbitrary state preparation:

```
from qiskit import QuantumCircuit
from qiskit.circuit.library import StatePreparation
import numpy as np

# Define the desired state vector
desired_state = [0.5, 0.5, 0.5, 0.5]

# Normalize the vector
normalized_state = desired_state / np.linalg.norm(desired_state)

# Create a quantum circuit with 2 qubits
qc = QuantumCircuit(2)

# Use StatePreparation to prepare the desired state
state_preparation = StatePreparation(normalized_state)
qc.append(state_preparation, [0, 1])
```

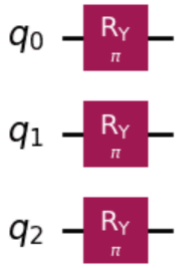
Angle encoding

Angle encoding makes use of rotation gates to encode information of x. The classical information defines what angles we will apply to the rotation gates:

$$|x\rangle = \otimes R(x_i)\,|0^n\rangle$$

Any of the rotation gates, R_x, R_y, or R_z, can be used in this operation. Typically, the number of qubits needed is equal to the dimension of the vector x. For example, if $x = [\pi, \pi, \pi]$, this would be encoded around the Y-axis if we choose R_y, corresponding to the state $|111\rangle$. We can construct this circuit and see the final state:

```
import numpy as np
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector

# Number of qubits = length of the classical information
n = 3

angle_enc = QuantumCircuit(n)

# X is the classical information
x = [np.pi, np.pi, np.pi]

# Add a layer of rotation y gates
for i in range(len(x)):
    angle_enc.ry(x[i], i)

state = Statevector(angle_enc)
roundstate = state.to_dict(decimals=3)
roundstate
```



The result would be:

```
{np.str_('111'): np.complex128(1+0j)}
```

Effectively coding the state 111 using angles.

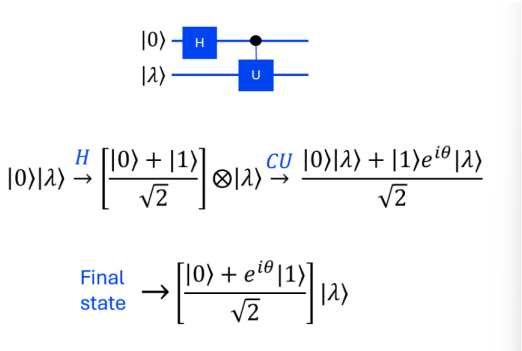
Phase encoding

Phase encoding involves embedding data into the phase of quantum states. This technique is effective for representing data in a way that directly influences quantum operations. For a dataset, the corresponding quantum state can be represented as:

$$|\psi\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} e^{ix_i|i\rangle}$$

In this equation, x_i each data element is encoded into the phase of the quantum state using the exponential function $e^{i\phi}$. The normalisation factor ensures that the quantum state is properly normalised.

This type of encoding comes up especially in QPE. There, we implement the phase encoding using a controlled unitary gate to apply phase kickback. This allows the state that we want to represent to be applied to the circuit as a phase instead of being applied into the basis. This process can be expanded further to apply multiple unitary gates, each encoding a phase from the classical data set.



IQP style encoding

IQP (Instantaneous Quantum Polynomial) style encoding is an advanced method that combines the principles of different encoding techniques to optimise data representation for specific quantum tasks. This method focuses on creating a balanced and efficient representation of data by encoding the data into the following state:

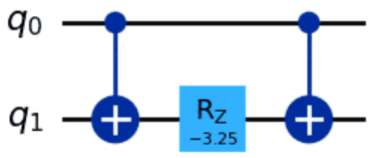
$$|x\rangle = (U_z(x)H^{\otimes n})^r |0^n\rangle$$

Where r is the circuit depth, which we repeat by iteration. The gate $U(x)$ is the key to encoding with IQP style, defined as:

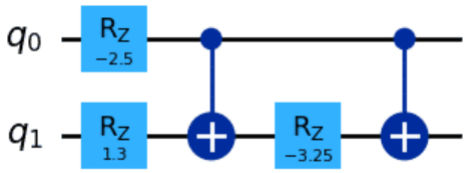
$$U_z(x) = \prod_{[i,j]\in S} R_{Z_iZ_j}(x_i x_j) \bigotimes_{k=1}^{n} R_z(x_k)$$

Where S is the set containing all pairs of qubits to be entangled using ZZ gates.

First, let's consider the U_{ZZ} gate. Its mathematical form can be seen as a two-qubit rotation gate around ZZ, which we can implement as:

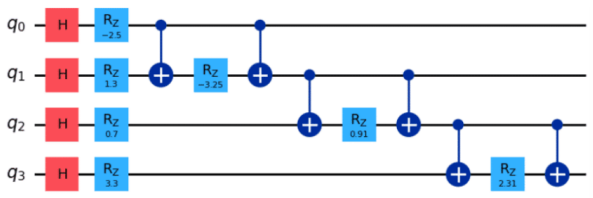


The values that the ZZ gate takes in are the multiplication of two classical values in our dataset corresponding to the values assigned to the qubits we are entangling. Continuing with the gate in the other part $U(x)$, for each qubit we apply R_z with one of the values of our classical data set, like so:




```
from qiskit import QuantumCircuit

# Number of qubits
n = 4
qc = QuantumCircuit(n)

x = [-2.5, 1.3, 0.7, 3.3]  # Classical information vector
S = [[0, 1], [1, 2], [2, 3]]  # The list containing all pairs to be entangled
r = 1  # The amount of times to repeat U(x)

for _ in range(r):
    # Hadamard layer
    qc.h(range(n))

    # Parametric Rz rotations
    for qubit in range(n):
        qc.rz(x[qubit], qubit)

    # Entanglement layer (ZZ gates via CNOT sandwich)
```

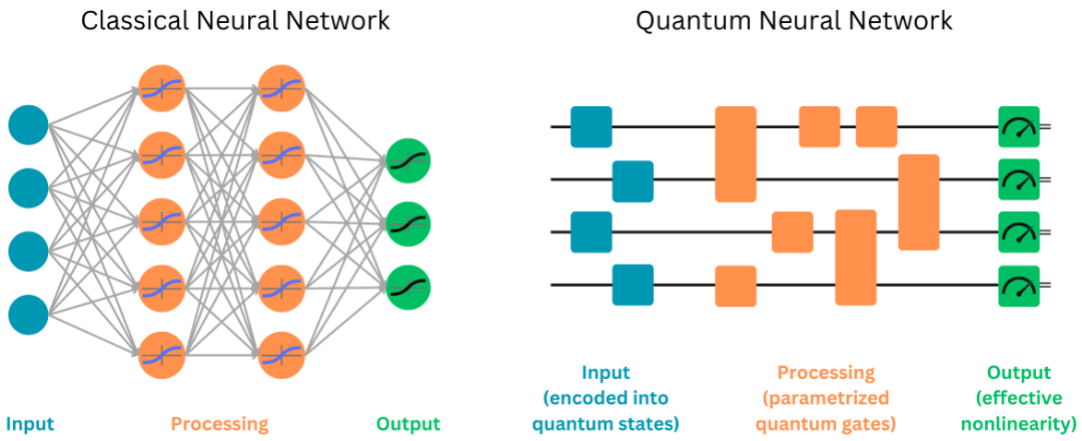
```
    for pair in S:
        control, target = pair
        qc.cx(control, target)
        qc.rz(x[control] * x[target], target)
        qc.cx(control, target)

qc.draw('mpl')
```



Importance of encoding techniques

Quantum encoding techniques serve as the foundational bridge between classical data and quantum computational advantage, determining the feasibility and efficiency of machine learning applications in quantum systems. The choice of encoding strategy directly impacts circuit depth, error resilience, and the ability to harness exponential quantum state spaces for tasks like classification. Crucially, tailored encodings that preserve domain-specific structures, such as molecular symmetries in chemistry, demonstrate how intelligent encoding design can inherently suppress errors and reduce resource demands. As quantum hardware evolves, the strategic selection and optimisation of encoding protocols will remain pivotal for unlocking scalable, accurate quantum machine learning solutions across industries.

M9: Quantum Neural Networks



1.1 Classical vs Quantum Neural Networks

Classical neural networks have become a cornerstone of modern machine learning. They consist of layers of interconnected nodes ("neurons") with adjustable parameters ("weights"), combined with nonlinear activation functions. The nonlinearity is crucial: without it, classical neural networks would effectively fall into a single linear transformation.

By contrast, quantum gates—the fundamental operations in quantum computing—are linear (unitary) transformations on quantum states. This means that if we restrict ourselves solely to unitary gates, we do not obtain the usual nonlinear activations of classical networks. However, measurement in quantum mechanics introduces a form of nonlinearity (probabilistic collapse of the wavefunction), and this plays an essential role in building neural-network-like architectures on quantum hardware.

A helpful way to see the analogy with classical neural networks is:

- **Data Input:** Just as classical networks take vectors of numbers, quantum neural networks receive data in the form of classical vectors that must be encoded into quantum states (e.g., amplitude encoding, basis encoding, or rotation encoding).
- **Network Processing:** In place of layers of classical neurons, QNNs use sequences (layers) of parametrized quantum gates. These gates are controlled by tunable parameters—akin to the trainable weights in classical networks.
- **Output:** The final measurement of qubits yields a classical number or set of numbers. Because of measurement, a QNN can exhibit effective nonlinearity in its overall input-output mapping, even though all intermediate gates are linear.

The Linear vs. Nonlinear Perspective in QNNs

From a purely mathematical standpoint, any single quantum gate (or a product of such gates) corresponds to a linear operator $U(\theta)$ acting on a Hilbert space. Consequently, if we never performed a measurement, the overall transformation from an initial state to a final state would remain linear in terms of the quantum amplitudes. In classical neural networks, by contrast, there is a built-in element of nonlinearity through activation functions like ReLU or sigmoid. Without that nonlinearity, multiple layers of linear operations would collapse into a single equivalent linear transformation, thus no expressive power is gained from stacking those layers.

In Quantum Neural Networks, however, measurement injects an effective nonlinearity. Upon measurement, the wavefunction collapses probabilistically, providing an output that does not vary linearly with respect to the input amplitudes. For instance, consider the expectation value of some observable O:

$$\langle O \rangle(\theta) = \langle 0|U^\dagger(\theta) O U(\theta)|0\rangle$$

Although $U(\theta)$ itself is linear, the mapping from the parameters θ to $\langle O \rangle(\theta)$ can exhibit effective nonlinearity because of how amplitudes and probabilities interact via the Born rule. This behavior means we can construct multi-layer analogues, where each layer is a sequence of quantum gates, and the final readout produces a value used in a cost or loss function.

Moreover, some proposals for QNNs incorporate intermediate measurements, partial resets, or classical feedback loops to emulate layer-by-layer .ªctivation.İn such architectures, partial measurement outcomes can condition subsequent gates, leading to even more pronounced effective nonlinearity. Although these techniques can be hardware-intensive, they are conceptually akin to injecting classical neural-network-like nonlinear activations into an otherwise linear quantum circuit.

Lastly, the interplay between data encoding and measurement can also yield nonlinear mappings in the classical data space. For example, if a data point x is encoded via parameterized rotations $R_y(2x_i)$, and subsequent entangling gates plus measurements are used, the final outcome is a function $f(x, \theta)$ that can be significantly more complex than a linear function of x. This underpins the idea that QNNs—despite each gate being a linear map on amplitudes—offer a form of nonlinearity once we consider how measured probabilities relate back to classical inputs and outputs.

1.2 Basic Components of QNNs

A typical quantum neural network for supervised or unsupervised tasks can be broken down into three major components:

Quantum Data Encoding

Before a quantum circuit can process classical data, the data $x \in \mathbb{R}^d$ must be mapped to a quantum state $|\psi(x)\rangle$. This feature map step is essential: different encoding methods can drastically alter a QNN's performance. Common approaches are:

- **Amplitude encoding:** encodes data into the amplitudes of a multi-qubit quantum state.
- **Basis encoding:** uses the computational basis states to represent data.
- **Rotation encoding:** employs parameterized single-qubit rotations (e.g., $R_y(2x)$ gates).

One of the main challenges is that amplitude encoding requires the normalization of data vectors and is typically used for lower-dimensional inputs because of the need to represent all components of x in the quantum amplitudes. Another strategy, known as data re-uploading (or data re-embedding), involves repeatedly encoding the same input data between layers of a variational circuit, allowing the network to capture more complex decision boundaries than a single round of encoding. In practice, one might also combine rotation encoding with controlled operations, or use partial amplitude/basis encoding depending on the nature of the dataset and the number of qubits available.

Parametrized Quantum Gates

After encoding, one applies a parametrized quantum circuit whose unitary operation $U(\theta)$ depends on a set of classical parameters θ. These gates constitute the learnable part of the QNN:

$$|\psi_{\text{out}}(x,\theta)\rangle = U(\theta)|\psi(x)\rangle$$

In many designs, these gates are arranged in "layers,.[ea]ch typically composed of single-qubit rotations and entangling operations (e.g., controlled-NOTs). This layered structure mirrors the hierarchical arrangement of neurons in classical networks.

A common approach, known as the two-local (hardware-efficient) ansatz (an initial guess or proposed functional form for a quantum circuit), relies on repeated blocks of single-qubit rotations and entangling gates (such as $CNOT$ or CZ), each considered one "layer."The chosen entangling pattern—linear, cyclic, or all-to-all—can strongly influence the expressivity and trainability of the QNN. Some QNNs make use of a tree-like connectivity (often referred to as a "tree tensor"structure) or other specialized ansätze to reduce circuit depth or to handle specific tasks, such as binary classification. However, deep circuits with random parameter initialization can lead to vanishing gradients (barren plateaus). By carefully adjusting the ansatz depth, choosing suitable parameter initializations, or leveraging problem-inspired configurations, it is possible to mitigate these optimization difficulties and enhance the training process.

Quantum Measurement

Finally, a measurement transforms the output state $|\psi_{\text{out}}\rangle$ back into classical information. For instance, measuring the expectation value of a Pauli operator on one or more qubits can provide a scalar output:

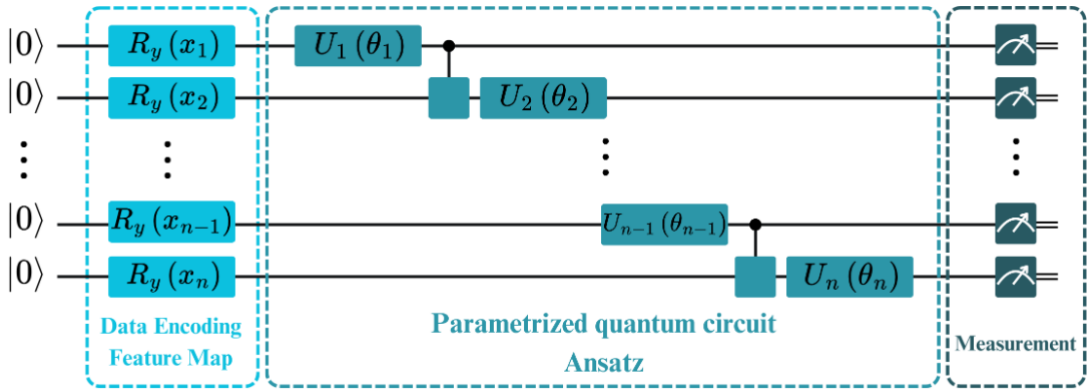
$$\langle O\rangle_{\text{out}} = \langle\psi_{\text{out}}|O|\psi_{\text{out}}\rangle$$

This is often used for classification (e.g., thresholding an expectation value) or for regression tasks (using it as a continuous prediction).

In some settings, one may measure multiple qubits or multiple observables to produce a vector of outputs, analogous to multi-class classification or multi-dimensional regression tasks. If these observables do not commute, one typically needs separate runs of the circuit to estimate each expectation value, increasing sampling requirements but allowing for richer output representations. Moreover, certain QNNs are designed to output a probability distribution over measurement outcomes—common in generative modeling—where the output probabilities are given by the Born rule,

$$p(z) = ||\langle z|\psi_{\text{out}}\rangle||^2$$

which can then be used for sampling or compared directly to classical data for training.



Motivation for Using QNNs Although the ultimate potential of QNNs is still an active research question, there are several reasons why quantum neural networks are of great interest:

Processing Quantum Data: When data itself is naturally quantum (e.g., outputs of quantum experiments or other quantum processes), it is often more efficient to keep it in a quantum representation. QNNs can learn directly from quantum states without costly classical readout.

High-Dimensional Feature Spaces: Quantum states live in exponentially large Hilbert spaces. A suitably chosen encoding scheme might allow QNNs to express complex functions that are difficult for classical networks to replicate.

Potential for Speedups: While not guaranteed for every problem, certain tasks may see quantum speedup or more efficient training and evaluation, especially if the hardware can exploit parallelism in quantum states.

Avoiding Overfitting: Some works have observed that certain variational circuits can resist overfitting more naturally than classical models of a similar size, though this is still an active area of research.

Roadmap of QNN Architectures In this module, we will explore several QNN architectures, each highlighting different ways of leveraging quantum effects for machine learning tasks:

Quantum Circuit Born Machines (QCBMs) Generative models that learn a probability distribution over measurement outcomes using the Born rule. They are often used for sampling tasks.

Quantum Feedforward Neural Networks (QFNNs) Networks that adapt classical feedforward ideas into a quantum framework, using layered parametrized circuits and measurement-based outputs.

Quantum Convolutional Neural Networks (QCNNs) An adaptation of classical CNNs to exploit local connectivity and pooling ideas in a quantum setting, aiming to reduce circuit depth while extracting features via entanglement.

Quantum Generative Adversarial Networks (QGANs) An advanced generative model where a quantum generator and a quantum or classical discriminator engage in an adversarial game to synthesize realistic data.

Quantum Autoencoders and Quantum Recurrent Neural Networks (Optional)

Quantum Autoencoders focus on compressing quantum data into fewer qubits, useful for noise reduction and dimensionality reduction. Quantum Recurrent Neural Networks extend the idea of recurrent architectures to the quantum domain, enabling sequence modeling with quantum states. Understanding these architectures will equip you with a broad toolkit to tackle both quantum-native and classical datasets in a quantum-enhanced fashion.

Conclusion Quantum Neural Networks bring together the principles of quantum computing and neural networks, introducing novel ways to handle data and potentially solve certain problems more efficiently or in ways unattainable with purely classical methods. While the field is still rapidly evolving—both theoretically and experimentally—learning the fundamentals of QNNs will prepare you to explore this cutting-edge domain where quantum information meets machine learning.

In the upcoming sections, we will deepen our understanding of each QNN architecture, looking at how to implement and train them in practice, and investigating some of the algorithmic advantages and challenges they offer.

Stay tuned as we start our journey into designing and training the next generation of neural networks—quantum neural networks.

Desing and training

2.1 Building a Quantum Neural Network

1. Data Preparation (Feature Map)

A QNN begins with a feature map, which takes classical input data $\vec{x}$ and encodes it in a quantum state. This is analogous to any preprocessing stage in classical neural networks but must respect the constraints of quantum computing. Common feature maps include:

- **Angle/Rotation Encoding:** Applying single-qubit rotation gates (e.g., $R_y(2x_i)$) to embed each data component into a qubit's angles.
- **Amplitude Encoding:** Mapping data vector elements to the amplitudes of a multi-qubit state, which requires normalization of the data.
- **Basis Encoding:** Using computational basis states to represent discrete features or categories.

2. Variational Circuit (Processing)

After encoding, the network applies a variational form (or ansatz) with learnable parameters θ. This circuit is designed to "transform" the encoded state in a way reminiscent of how layers in a classical neural network transform the activations. The QNN's variational form may be structured in layers, each consisting of:

- **Parametric Single-Qubit Rotations:** For example, $R_y(\theta)$ or $R_z(\phi)$.
- **Entangling Gates:** Such as controlled-NOT (CNOT), controlled-Z, or other two-qubit operations, arranged in a pattern (linear, cyclic, or all-to-all) to introduce entanglement between qubits.

In essence, feature maps and variational forms are both variational circuits, but they differ in purpose. Feature maps depend on the input data, while the variational form depends on the optimizable parameters θ.

3. Measurement and Output

Finally, measurements map the quantum state back to a classical output. Depending on the QNN's goal—e.g., binary classification, multi-class classification, or regression—one might measure:

- The expectation value of a certain operator (e.g., measuring one qubit or the parity of multiple qubits).
- A probability distribution over multiple measurement outcomes (e.g., for sampling or generative models).

Choosing the measurement operator carefully is crucial, as it directly determines how the model's output is read and used in a loss function. **2.1 Building a Quantum Neural Network**

1. Data Preparation (Feature Map)

A QNN begins with a feature map, which takes classical input data $\vec{x}$ and encodes it in a quantum state. This is analogous to any preprocessing stage in classical neural

networks but must respect the constraints of quantum computing. Common feature maps include:

Angle/Rotation Encoding: Applying single-qubit rotation gates (e.g., $R_y(2x_i)$) to embed each data component into a qubit's angles.

Amplitude Encoding: Mapping data vector elements to the amplitudes of a multi-qubit state, which requires normalization of the data.

Basis Encoding: Using computational basis states to represent discrete features or categories.

2. Variational Circuit (Processing)

After encoding, the network applies a variational form (or ansatz) with learnable parameters θ. This circuit is designed to "transform" the encoded state in a way reminiscent of how layers in a classical neural network transform the activations. The QNN's variational form may be structured in layers, each consisting of:

Parametric Single-Qubit Rotations: For example, $R_y(\theta)$ or $R_z(\phi)$.

Entangling Gates: Such as controlled-NOT (CNOT), controlled-Z, or other two-qubit operations, arranged in a pattern (linear, cyclic, all-to-all) to introduce entanglement between qubits.

In essence, feature maps and variational forms are both variational circuits, but they differ in purpose. Feature maps depend on the input data, while the variational form depends on the optimizable parameters θ.

3. Measurement and Output

Finally, measurements map the quantum state back to a classical output. Depending on the QNN's goal—e.g., binary classification, multi-class classification, or regression—one might measure:

The expectation value of a certain operator: for example, measuring one qubit or the parity of multiple qubits.

A probability distribution over multiple measurement outcomes: for instance, for sampling or generative models.

Choosing the measurement operator carefully is crucial, as it directly determines how the model's output is read and used in a loss function.

2.2 Training a QNN

Training a QNN typically means adjusting the parameters $\boldsymbol{\theta}$ in the variational circuit to minimize a loss function $L(\boldsymbol{\theta})$. This process takes inspiration from classical neural network optimization yet incorporates additional challenges.

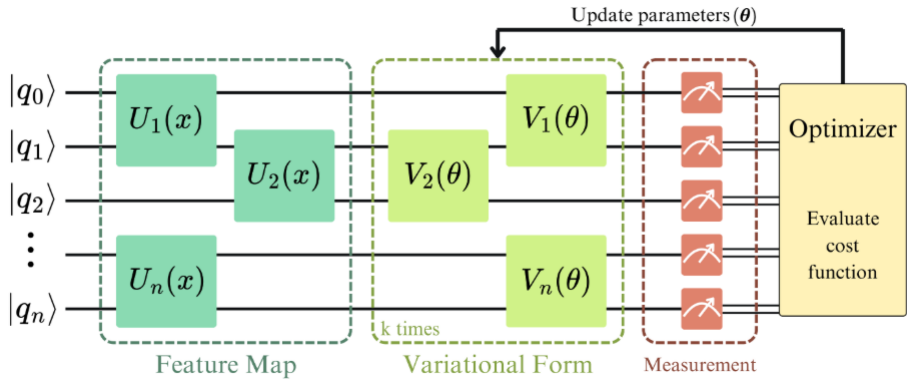
Assume that a QNN is specified on n quantum registers with l layers of adjustable quantum gates, where each adjustable gate is controlled by a single parameter $(\theta_i^k)_{i=1,\ldots,n;\, k=1,\ldots,l}$. In this case, $\boldsymbol{\theta} \in M_{n,l}$ is an $n \times l$ matrix of adjustable network parameters:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1^1 & \cdots & \theta_l^1 \\ \vdots & \ddots & \vdots \\ \theta_1^n & \cdots & \theta_l^n \end{bmatrix}$$

Recall that a QNN takes an input (a quantum state that encodes a sample from the dataset), applies a sequence of quantum gates (the parameterised quantum circuit controlled by at most $n \times l$ adjustable parameters), and performs the measurement of an observable M on the chosen quantum register. An example of an observable is the Pauli-Z gate, and the result of a single measurement is ± 1 for a qubit found in the state $|0\rangle$ or $|1\rangle$, respectively. The value of the measured observable is mapped into a value of a binary

variable $\{0, 1\}$.

This process is repeated N times for each sample in order to collect sufficient statistics for the classification result.



Defining the Loss Function

The first step in finding an optimal configuration of adjustable parameters $\boldsymbol{\theta}$ is to choose an appropriate cost function — an objective function that represents the total error in classifying samples from the training dataset and which can be minimized by changing the adjustable network parameters.

Let $\mathbf{y} = (y_1, \ldots, y_M)$ be a vector of labels and $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}) = (f(x_1, \boldsymbol{\theta}), \ldots, f(x_M, \boldsymbol{\theta}))$ a vector of predictions for the training dataset consisting of M samples.

For supervised tasks (e.g., classification), one might use a mean-squared error or cross-entropy loss:

$$L(\boldsymbol{\theta}) = \frac{1}{M} \sum_{k=1}^{M} (y_k - f(x_k, \boldsymbol{\theta}))^2$$

where y_k is the true label and $f(x_k, \boldsymbol{\theta})$ is the QNN's output (for instance, the measured expectation value). Unsupervised tasks, like generative modeling, often define losses based on probability distributions or fidelity.

The next step is an iterative update of the adjustable parameters in the direction that reduces the value of the cost function by using an optimization method, for example, getting this direction by calculating the cost function gradient.

—

Gradient-Based Approaches

Finite Difference Scheme. One straightforward method for computing gradients, especially in smaller QNNs, is to approximate partial derivatives numerically:

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i^j} \approx \frac{L(\ldots, \theta_i^j + \Delta\theta_i^j, \ldots) - L(\ldots, \theta_i^j - \Delta\theta_i^j, \ldots)}{2\Delta\theta_i^j}$$

This method can be effective but may suffer from truncation errors and constraints on how small $\Delta\theta$ can be in noisy hardware environments.

Analytic Gradient and Parameter-Shift Rule. A more precise approach uses analytic gradients for one-qubit gates. The parameter-shift rule states that if a gate is generated by a Pauli operator (e.g., $R_x(\theta)$, $R_y(\theta)$), its partial derivative can be computed by shifting θ by $\pm\pi/2$. Concretely,

$$\frac{\partial}{\partial \theta_j} \langle O \rangle(\boldsymbol{\theta}) = \frac{1}{2} \Big[\langle O \rangle(\boldsymbol{\theta} + \tfrac{\pi}{2} e_j) - \langle O \rangle(\boldsymbol{\theta} - \tfrac{\pi}{2} e_j) \Big]$$

This typically provides more stable gradient estimates than finite difference, particularly on real quantum devices that have limited rotational precision.

Iterative Update (Gradient Descent). Regardless of how gradients are obtained, the parameters are updated towards the direction of the steepest descent of the cost function. At step $u + 1$, we update the system to

$$\theta_i^j \leftarrow \theta_i^j - \eta \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i^j}$$

for each $i = 1, \ldots, n$, $j = 1, \ldots, l$, and η is the learning rate. Modern optimizers like Adam or RMSProp can also be used to handle adaptive learning rates.

—

Gradient-Free Approaches

Not all QNN settings lend themselves easily to gradient-based methods (e.g., complex cost landscapes, hardware precision limits, or certain ansätze). Particle Swarm Optimization (PSO) is an example of a powerful gradient-free search heuristic.

In the standard PSO formulation, a number of particles are placed in the solution space of some problem and each evaluates the fitness at its current location. Each particle then determines its movement through the solution space by combining some aspects of the history of its own fitness values with those of one or more members of the swarm, and then moves through the solution space with a velocity determined by the locations and processed fitness values of those other members, along with some random perturbations.

The standard procedure follows three steps:

1. Initialization: Randomly place particles (candidate solutions) in the parameter space. 2. Velocity Update: Each particle's velocity depends on momentum, its own best-known position, and the global best position found by the swarm. 3. Position Update: Each iteration, particles move through parameter space aiming to reduce the cost.

PSO and other evolutionary algorithms (like genetic algorithms or Nelder–Mead) can be especially beneficial if the cost function is non-convex or the gradient is difficult to approximate reliably.

—

Considerations in QNN Training

- *Gradient-Free Methods:* When parameter-shift is infeasible or the landscape is too rugged, methods such as evolutionary strategies, Nelder–Mead, or particle swarm optimization can be applied. - *Sampling Noise:* Real hardware requires multiple measurement "shots," introducing noise into gradient estimates. Increasing shots reduces variance but raises runtime costs. - *Barren Plateaus:* Deep QNNs can encounter exponentially vanishing gradients, necessitating careful initialization, circuit design, or problem-specific ansätze to preserve useful gradient magnitudes. - *Classical–Quantum Feedback Loops:* Typically, the quantum device (or simulator) evaluates the circuit for given $\boldsymbol{\theta}$, returns measured values, and a classical optimizer adjusts parameters before the next iteration.

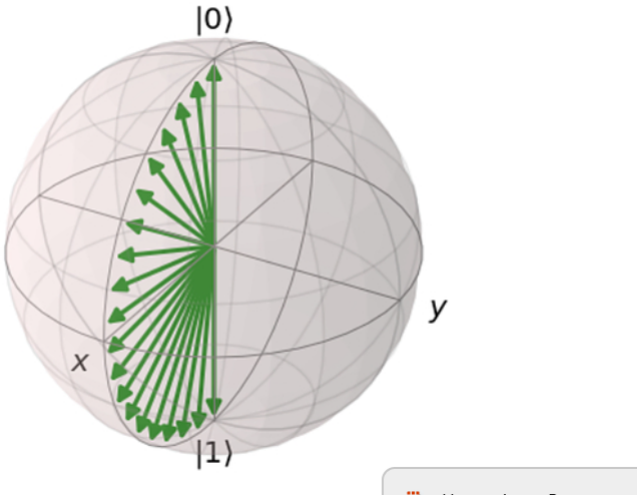
—

Simple Example

Let's look at an example of a very small QNN capable of optimizing a single parameter that defines a quantum state of a qubit.

The objective will be to find the degrees to rotate a qubit that starts in the state $|0\rangle$ so that it reaches the state $|1\rangle$, making a rotation around the Y axis.

For this example we will be using Qiskit, so we need to install it:




```
!pip install qiskit[visualization]==1.4.2
```

Let's define a cost function (or loss function), which will check how similar the quantum circuit measurement is to state 1, which is our target state:

```
def cost_state_1(prob_meas_state_1):
    return 1 - prob_meas_state_1
```

Now, to calculate the state measured by our quantum circuit, we must obtain the average of several measurements:

```
from qiskit import QuantumCircuit
import qiskit.quantum_info as qi
from math import radians, degrees

def average_cost(angle, fn_costo, shots=2000):
    qc = QuantumCircuit(1)
    qc.ry(radians(angle[0]), 0)
    sv = qi.Statevector(qc)
    counts = sv.sample_counts(shots=shots)
    average_state_1 = (counts['1'] if '1' in counts else 0)/shots
    average_cost = fn_costo(average_state_1)
    print('Try: angle =', angle, 'produces an average cost =', average_cost)
    return average_cost
```

To carry out the optimization process, we will use a classical optimizer from `scipy`:

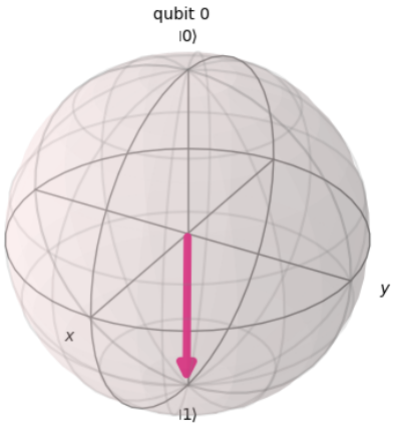
```
from scipy.optimize import minimize

initial_guess = 90

result = minimize(average_cost,
                  initial_guess,
                  args=(cost_state_1,),
                  method='Powell')
```

After running, the optimizer finds an angle close to 180°, yielding a cost ≈ 0, meaning the qubit was rotated from $|0\rangle$ to $|1\rangle$.

```
print('\nOptimized angle in degrees:', [a % 360 for a in result.x])
```

Finally, we can visualize the state on the Bloch sphere:

```
from qiskit.visualization import plot_bloch_multivector

qc = QuantumCircuit(1)
qc.ry(radians(result.x[0]), 0)
sv = qi.Statevector(qc)
plot_bloch_multivector(sv)
```



Conclusions

Designing and training a quantum neural network requires careful consideration of data encoding, the choice of variational ansatz, and measurement schemes. While core gradient-descent ideas from classical machine learning carry over to QNNs, special techniques like parameter-shift or gradient-free algorithms (PSO, evolutionary methods) often become necessary in the face of hardware noise, limited qubit connectivity, and inherently high-dimensional state spaces.

From a practical standpoint, the decision between finite difference, analytic gradient, or gradient-free methods hinges on the hardware constraints, circuit depth, and cost function complexity. Likewise, the performance of QNN classifiers on real-world datasets (e.g., credit approval tasks) can be enhanced through techniques like ensemble learning or judicious choice of circuit architecture—mirroring many best practices in classical ML while highlighting quantum-specific challenges such as barren plateaus and shot noise.

By striking a balance between circuit complexity and hardware feasibility, leveraging specialized optimization schemes, and potentially combining QNNs with classical ensemble strategies, practitioners can unlock the promising capabilities of QNNs for both quantum-native and classical data applications. As quantum hardware evolves, so too will the range of viable QNN architectures, training methodologies, and real-world problem domains ripe for quantum advantage.

Quantum circuit born machines (QCBMS)

3.1 Overview and Motivation

Classical generative models, like Restricted Boltzmann Machines (RBMs) or certain neural architectures (e.g., Generative Adversarial Networks (GANs)), have long been used for tasks such as creating synthetic data and anonymization. Quantum Circuit Born Machines (QCBMs) can be viewed as the quantum analog of RBMs, where each qubit effectively corresponds to a visible binary unit. They offer:

- **Potentially Higher Expressive Power:** Research suggests that with only polynomially many parameters, QCBMs can represent distributions that classical RBMs cannot easily capture.
- **Fast Sampling:** Once parameters are set, generating a sample requires a single execution (one circuit run plus measurement), compared to the iterative Gibbs sampling required in some classical models.
- **Quantum Data Loading:** QCBMs can directly embed quantum states, potentially enabling further quantum processing steps.

3.2 QCBM Architecture

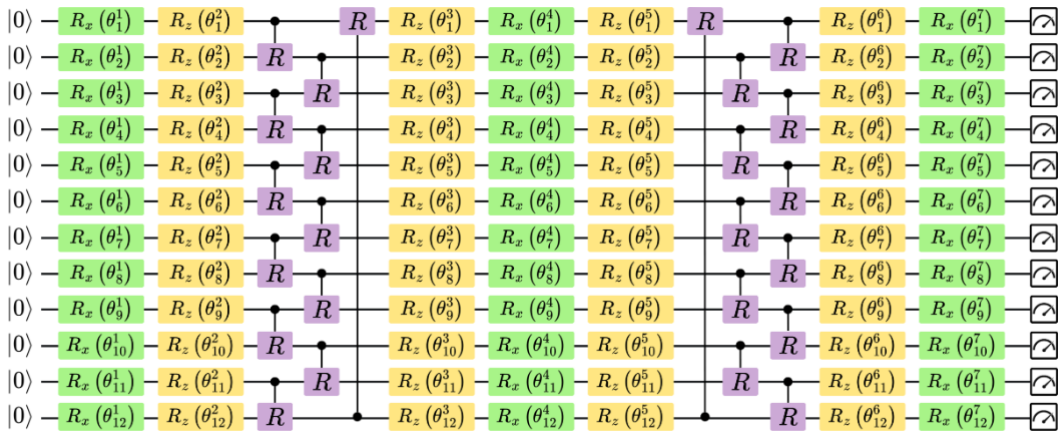
A Quantum Circuit Born Machine (QCBM) starts with all qubits in the state $|0\rangle$ and processes them through a sequence of one-qubit gates (parametric rotations) and fixed two-qubit gates (like CNOT or CPhase). After the final layer, a measurement in the computational basis yields a bitstring sample. The distribution of these bitstrings is the model's learned probability distribution.

Example Layout

A typical QCBM might interleave:

- **Adjustable Single-Qubit Gates:** Rotations around x, y, or z axes, each parameterized by a real θ.
- **Fixed Two-Qubit Gates:** For example, a ring or line of controlled rotations (CNOT, CPhase, etc.).
- **Measurements:** At the end, measuring in the Z basis gives an n-bit sample (if using n qubits).

This structure can be repeated in multiple layers, forming a deeper circuit. As with quantum neural networks, there is an interplay between circuit depth, hardware connectivity, and the risk of encountering barren plateaus.



The figure above shows a 12-qubit QCBM with two layers of controlled rotation gates $R = R_G(\phi)$ for $G \in \{X, Y, Z\}$ and $\phi \in [-\pi, \pi]$, where G and ϕ are fixed, and three layers of one-qubit gates $R_X(\theta)$ and $R_Z(\theta)$ with a total of seven adjustable gates per quantum register. The circuit is wide enough and deep enough to learn a complex distribution of a continuous random variable while remaining implementable on existing NISQ devices: the 12-digit binary representation of a continuous random variable provides sufficient precision and seven adjustable parameters (rotation angles) per qubit provide sufficient flexibility.

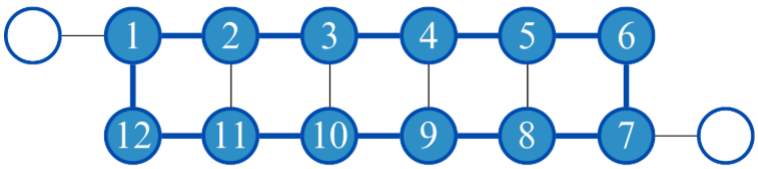
Hardware-Efficient Embeddings

Because near-term quantum devices have limited qubit connectivity and native gate sets, QCBM designs often aim to be hardware-efficient:

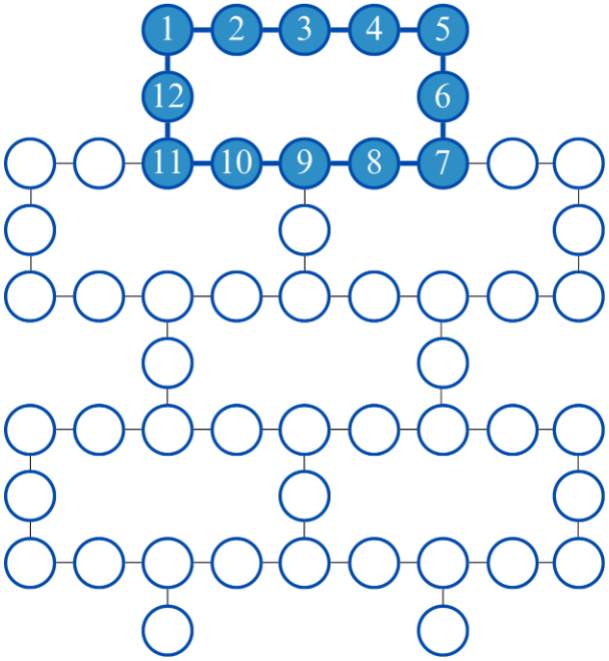
- **Connectivity:** Place qubits so that two-qubit gates match the device's coupling map (e.g., a ring or linear chain).
- **Native Gate Set:** Use the device's built-in operations (e.g., IBM's cross-resonance leading to CNOT).

Such an approach ensures minimal overhead in gate decomposition and can improve fidelity on noisy intermediate-scale quantum (NISQ) hardware.

Let's look at the quantum circuit proposed in the previous figure. We observe that it requires sequential qubit connectivity where qubit n is directly connected with qubits $n-1$ and $n+1$ but does not have to be directly connected with other qubits. This architecture can, for example, be supported by IBM's Melbourne system (currently no longer available), shown below, where the 12 shaded qubits correspond to the 12 quantum registers in the quantum circuit. The thick lines represent connections used in the QCBM ansatz, while the thin lines represent all other available qubit connections.



The 53-qubit IBM's Rochester system (currently no longer available), shown below, can also be used to implement this QCBM architecture. Here, we have several choices for embedding the QCBM circuit (12 linearly connected qubits forming a closed loop); shaded qubits show one such possibility.



3.3 Differentiable Training of QCBMs

Training a QCBM means finding parameters θ that make the model's output distribution $p_\theta(x)$ match some target distribution $\pi(x)$ derived from a classical dataset.

Data-to-Binary Conversion: QCBMs are inherently binary; any continuous data must be binarized—for instance, using an M-bit binary encoding for each dimension.

Cost Function: The objective is typically a measure of distance between $p_\theta(\cdot)$ and $\pi(\cdot)$. Examples include:

- **Total Variation Distance:** $\sum_x |p_\theta(x) - \pi(x)|$
- **Maximum Mean Discrepancy (MMD):** with a chosen kernel $K(x, y)$, such as a Gaussian mixture, often easier to optimize.
- **Quantum Kernels:** potential use when evaluating similarity in a quantum-native manner.

Gradient Descent:

- **Parameter-Shift Rule:** For gates generated by Pauli operators, gradient-based training is possible via parameter shifts, similar to QNN methods.
- **Automatic Differentiation:** On simulators, one may leverage backpropagation-like frameworks.

Like a quantum neural network, a QCBM can suffer from barren plateaus if the circuit is deep or randomly initialized, motivating alternative training schemes.

Cost Functions

The differentiable learning of QCBM follows the same principles as training QNNs: minimization of the cost function using the gradient descent method. The main difference lies in the form of the cost function. In the case of a QNN-based classifier, the cost represents classification error, while for a QCBM it represents the distance between two probability distributions: the distribution of samples in the training dataset and the one generated by the model.

Let θ denote the set of adjustable QCBM parameters, $p_\theta(\cdot)$ the QCBM distribution, and $\pi(\cdot)$ the data distribution. Then, the cost function $L(\theta)$ can be defined as:

$$L(\theta) = \sum_x |p_\theta(x) - \pi(x)|$$

where the sum runs over all samples x in the dataset. This cost function is a strong metric but may be difficult to handle. An efficient alternative is the Maximum Mean Discrepancy (MMD):

$$L(\theta) = \mathbb{E}_{x,y\sim p_\theta}[K(x,y)] - 2\,\mathbb{E}_{x\sim p_\theta, y\sim\pi}[K(x,y)] + \mathbb{E}_{x,y\sim\pi}[K(x,y)]$$

where $K(\cdot,\cdot)$ is a kernel function, i.e., a measure of similarity between points in the sample space. A popular choice is the Gaussian mixture kernel:

$$K(x,y) = \frac{1}{c}\sum_{i=1}^{c} \exp\left(-\frac{\|x-y\|^2}{2\sigma_i^2}\right)$$

for some $c \in \mathbb{N}$, where $(\sigma_i)_{i=1,\dots,c}$ are the bandwidth parameters of each Gaussian kernel and $\|\cdot\|$ denotes the L_2 norm.

We can also explore the possibility of using **quantum kernels**, which may provide an advantage over classical methods for kernels that are difficult to compute classically. For example, we can consider a non-variational quantum kernel method, which uses a quantum circuit $U(x)$ to map real data into a quantum state $|\phi(x)\rangle$ via a feature map:

$$|\phi(x)\rangle = U(x)\,|0\rangle^{\otimes n}$$

The kernel function is then defined as the squared inner product:

$$K(x,y) = |\langle\phi(x)|\phi(y)\rangle|^2$$

This quantum kernel is evaluated on a quantum computer and is generally hard to compute on a classical one. Taking into account the mapping $|\phi(x)\rangle = U(x)\,|0\rangle^{\otimes n}$ and denoting $|0\rangle = |0\rangle^{\otimes n}$, the kernel becomes:

$$K(x,y) = |\langle 0|U^\dagger(x)U(y)|0\rangle|^2$$

which corresponds to the probability of measuring the all-zero outcome. It can be computed by measuring, in the computational basis, the state that results from running the circuit $U(y)$ followed by $U^\dagger(x)$.

3.4 Non-Differentiable Approaches

Due to optimization challenges on NISQ hardware (noise, shot variance, large parameter spaces), evolutionary algorithms or other derivative-free methods can be effective alternatives for training QCBMs.

Genetic Algorithms (GA): GA is a powerful evolutionary search heuristic. It performs a multi-directional search by maintaining a population of proposed solutions (chromosomes) for a given problem. Each solution is represented in a fixed alphabet with an established meaning (genes). The population undergoes simulated evolution, where relatively good solutions produce offspring that subsequently replace worse ones. The quality of each solution is estimated with an objective function (environment).

For QCBMs, we can:

- Represent each QCBM parameter configuration as a çhromosome".
- Mutate angles randomly with decreasing mutation rates, balancing exploration vs.

exploitation.

- Evaluate a cost function (e.g., distribution mismatch) by generating samples from the QCBM.
- Select top performers to propagate to the next generation.

Particle Swarm Optimization (PSO): Another heuristic method, often discussed in the context of QNNs, but equally applicable to generative circuits like QCBMs. It uses a swarm of candidate solutions that adjust their positions based on both individual and collective performance, gradually converging toward optimal parameters.

Such methods bypass the need for stable gradients but can require many function evaluations. They are particularly suitable for noisy intermediate-scale quantum (NISQ) devices, where gradient estimation may be unreliable or computationally expensive. **3.5 Comparisons with Classical RBMs**

Classically, Restricted Boltzmann Machines (RBMs) capture probability distributions via a bipartite graph connecting visible and hidden binary units. Quantum Circuit Born Machines (QCBMs) can be viewed as their quantum counterparts.

Key Comparisons:

- **Number of Parameters:** For example, an RBM with 12 visible units and 7 hidden units corresponds to a QCBM with 12 qubits and 7 single-qubit rotations per qubit.
- **Sampling:** RBMs require iterative Gibbs sampling, whereas QCBMs produce samples in one shot through quantum measurement.
- **Performance:** Empirical studies [**?**, **?**, **?**, **?**] show that QCBMs can match or sometimes outperform RBMs for certain distributions, although a definitive quantum advantage remains an open question.

In finance, both RBMs and QCBMs can model joint distributions of market risk factors or price returns. The QCBM, in particular, might learn heavy-tailed or multi-modal return distributions, generating synthetic data for scenario analysis, stress testing, or improved risk management.

Additionally, similar principles extend beyond finance to other domains:

- **Medical Imaging:** Generating synthetic scans for data augmentation or anomaly detection without exposing sensitive data.
- **Material Science:** Sampling candidate atomic configurations to accelerate the discovery of new compounds.

Comparison Framework

The classical RBM is a generative model inspired by statistical physics, where the probability of a particular data sample v is given by the Boltzmann distribution:

$$P(v) = \frac{1}{Z} e^{-E(v)}$$

Here, $E(v)$ is the (positive) energy of the data sample (samples with lower energy have higher probability), and Z is the partition function, the normalization factor of the probability density:

$$Z = \sum_{v} e^{-E(v)}$$

Alternatively, the probabilistic nature of quantum mechanics allows us to model the probability distribution using a quantum state $|\psi\rangle$:

$$P(v) = \langle\psi|P_v^\dagger P_v|\psi\rangle$$

where P_v is a measurement operator (e.g., projection). Since the quantum state $|\psi\rangle$ is normalized, we have:

$$\langle\psi|\psi\rangle = 1$$

This approach is realized in the Quantum Circuit Born Machine (QCBM), where generative modeling of probability density is translated into learning a quantum state. The goal of the QCBM's parameterized circuit is to create a quantum state $|\psi\rangle$ that encodes the desired probability distribution, starting from the initial state $|0\rangle^{\otimes n}$, with sampling performed by applying the measurement operators.

Therefore, providing a classical benchmark for QCBMs consists of finding a suitable RBM configuration that allows comparison between two generative methods for the probability distribution $P(v)$: one derived from the classical RBM, and the other from the quantum QCBM. **3.8 Practical Tips and Conclusions**

Circuit Depth vs. Hardware: Strive to keep the QCBM circuit depth moderate to minimize decoherence and noise accumulation on NISQ devices.

Binarization Strategy: Tailor the number of bits M according to feature importance and available qubit budget.

Mutation Scheduling: When using Genetic Algorithms (GAs), start with a high mutation rate to explore the parameter space broadly, then gradually reduce it to refine local optima.

Distribution Metrics: Employ robust cost functions such as the Maximum Mean Discrepancy (MMD) or Total Variation Distance. Complement these with classical statistical tests—such as the Kolmogorov–Smirnov test or Q–Q plots—to validate the quality of the learned distribution.

Ensemble or Hybrid Approaches: Combining QCBMs with classical models, or ensembling multiple QCBM configurations, can improve sampling robustness and generalization.

In summary, Quantum Circuit Born Machines (QCBMs) provide a promising framework for generating complex probability distributions on quantum hardware. While large-scale quantum advantage is still under investigation, early benchmarks against classical Restricted Boltzmann Machines (RBMs)—particularly in finance—suggest that QCBMs can match or even surpass classical generative performance. Continued progress in both hardware reliability and algorithmic design is expected to further enhance their practical viability.

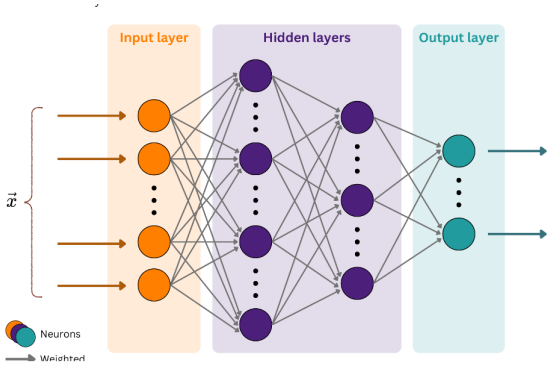
Quantum Feedforward Neural Networks (QFNNs)

4.1 Overview and Motivation

A classical feedforward neural network (often called a multilayer perceptron, or MLP) consists of an input layer that receives data x, one or more hidden layers that successively transform the data through linear operations and nonlinear activations, and an output layer that produces the network's final prediction (e.g., a class label or regression value). Each hidden layer typically computes:

$$z^{(\ell)} = \sigma\Big(W^{(\ell)} z^{(\ell-1)} + b^{(\ell)}\Big)$$

where $W^{(\ell)}$ and $b^{(\ell)}$ are learnable weights and biases, and σ is a nonlinear activation function (e.g., ReLU, sigmoid). $z^{(\ell)}$ represents the activation vector produced by the ℓ-th layer — in other words, the output of that layer, which becomes the input of the next layer $\ell + 1$. Each component of $z^{(\ell)}$ corresponds to the activation value of a "neuron.ºr unit of that layer.



Quantum Feedforward Neural Networks (QFNNs) replicate this architecture in a quantum setting by replacing the linear and nonlinear layers of a classical network with quantum circuits. Specifically:

- **Quantum Data Encoding:** Classical inputs $\vec{x}$ are embedded into an n-qubit quantum state via a feature map $U_F(\vec{x})$.
- **Variational Layers:** Instead of weight matrices and activation functions, QFNNs use parameterized quantum gates (e.g., single-qubit rotations, multi-qubit entangling operations) as trainable layers—represented by a variational form $U(\boldsymbol{\theta})$.
- **Quantum-to-Classical Readout:** Measurements of the final quantum state provide the network's outputs for tasks such as classification or regression.

One key difference is that each quantum gate is a linear operator; thus, the measurement process plays the role of introducing effective nonlinearity. This makes QFNNs an intriguing blend of quantum circuit design and machine learning principles, mirroring the structure of classical feedforward networks but executed on quantum hardware. **4.2 Hybrid vs. Coherent QFNNs**

A QFNN can be implemented in either a hybrid or a coherent manner:

- **Hybrid QFNN:** Each "neuron.ºr layer outputs a classical value (obtained by measuring an ancillary or target qubit). That output is then fed into the next layer, potentially re-encoded into a new quantum state. This approach is modular—each quantum layer is relatively small and can be executed separately on quantum hardware—but requires repeated measurements and re-encodings.
- **Coherent QFNN:** Qubits remain in a superposition across multiple layers without intermediate measurements. All the "neurons"(parameterized gates) are implemented simultaneously on the same multi-qubit register, preserving coherence between layers. This can reduce overhead from repeated measurements but tends to be more demanding on hardware, as it may require many qubits and deeper circuits.

Both approaches share the same conceptual goal: learn a function $f(\vec{x})$ by adjusting quantum gate parameters, but they differ in how classical outputs connect one layer to the next [**?**].

4.3 QFNN Architecture

Input Encoding: A QFNN typically starts with a feature map $U_F(\vec{x})$ that transforms $|0\rangle^{\otimes n}$ into an encoded state:

$$|\psi_x\rangle = U_F(\vec{x})|0\rangle^{\otimes n}$$

Common encoding methods include:

- **Rotation Encoding:** Use single-qubit rotations (R_x, R_y, R_z) whose angles depend on the features.
- **Amplitude Encoding:** Map normalized data vectors into amplitudes of a multi-qubit state.
- **Basis Encoding:** Represent discrete features directly in qubit basis states.

A more in-depth look at feature maps is in Section 2.1.1.

Parameterized Quantum Layers: After encoding, the network processes the state via one or more variational layers. Each layer can consist of:

- **Parametric Single-Qubit Rotations:** $R_\mu(\theta)$ gates on each qubit.
- **Entangling Gates:** Controlled-NOT (CNOT), controlled-phase, or other multi-qubit operations to introduce entanglement.

A typical QFNN may stack multiple such layers, each with its own set of trainable parameters θ. The final layer may or may not be followed by additional gates depending on the design.

A more detailed discussion of variational forms (or ansätze) appears in Section 2.1.2.

Measurement and Output: The QFNN's output is extracted by measuring the qubits. In a supervised learning scenario:

- For **binary classification**, measure a single qubit in the Z-basis, interpreting measurement outcomes (0 or 1) or expectation values $\langle Z \rangle$ as the network's prediction.
- For **multi-class or regression** tasks, measure multiple qubits or compute more complex observables to yield higher-dimensional outputs.

4.4 Quantum Activation Functions

A hallmark of classical feedforward networks is the nonlinear activation in each neuron. In quantum circuits, gates are linear, so nonlinearity arises chiefly from measurement or other non-unitary steps:

- **Measurement-Based Activations:** A qubit (or set of qubits) is measured, producing a classical result that can be re-encoded into another qubit for the next layer (hybrid approach). This partial measurement effectively inserts a nonlinearity.
- **Non-Linear Post-Processing:** Even if no intermediate measurement is performed, the final measurement outcome and subsequent classical post-processing can yield nonlinear mappings from input to output.
- **Coherent .ᴬctivations":** Some proposals attempt to implement approximate acti-

vation functions using controlled gates and ancillas, though these are more hardware-intensive and less common on near-term devices.

Regardless of implementation, effective nonlinearity is essential to ensure the QFNN can learn more than a simple linear function of the input state.

4.5 Supervised Learning with QFNNs

Classification and Regression: Similar to classical feedforward networks, QFNNs can handle:

- **Binary Classification:** Map x to a single-qubit measurement. Threshold the expectation value $\langle Z \rangle$, or interpret measurement outcomes as labels $\{0, 1\}$ or $\{-1, +1\}$.
- **Multi-Class:** Use multiple qubits or measurement operators to distinguish among more classes.
- **Regression:** Obtain a continuous output by measuring an expectation value (e.g., $\langle Z \otimes Z \rangle$) or by converting bitstring readouts into real-valued quantities.

Hybrid Optimization: Because each QFNN layer is a quantum circuit, the entire model's parameters θ can be trained using classical optimization methods, guided by:

- **Parameter-Shift Rule:** Compute gradients by shifting gate parameters θ by $\pm\frac{\pi}{2}$.
- **Automatic Differentiation:** For circuit simulators integrated with ML frameworks (e.g., PennyLane, TensorFlow Quantum).
- **Gradient-Free Methods:** If gradient computation is difficult (e.g., large circuits or hardware constraints), evolutionary or black-box optimizers can be applied.

A classical-quantum feedback loop is typically formed, where each iteration:

1. Sets parameters θ,
2. Runs the quantum circuit to compute the cost/loss,
3. Returns measurements to a classical optimizer,
4. Updates parameters and repeats.

4.6 QFNN Use Cases

Quantum-Enhanced Image Classification:

- **Data Encoding:** Convert images (or patches) into qubit states using rotation or amplitude encoding.
- **QFNN Training:** Variational layers and measurement operators classify images into categories (e.g., digits or medical anomalies).
- **Deployment:** Once trained, the circuit can classify new images rapidly, potentially exploiting quantum parallelism for scalability.

Combining classical pre-processing with quantum layers may allow QFNNs to detect subtle patterns and achieve accuracy comparable to classical networks, while benefiting from efficient sampling and high-dimensional Hilbert spaces.

Quantum Recommender Systems:

- **Data Encoding:** Represent user-item interactions as binary vectors on qubits via angle or basis encoding.
- **QFNN Training:** The QFNN learns to map user-item states to preference scores, adjusting gate parameters to minimize prediction error.
- **Prediction:** Measured outputs determine predicted ratings or relevance scores for new user-item combinations.

This approach can exploit quantum correlations to capture complex relationships, improving recommendation accuracy in domains like e-commerce or media streaming.

Quantum Control and Optimization:

- **Data Encoding:** Model a control problem (e.g., scheduling or resource allocation) as feature vectors describing system states.
- **QFNN Training:** The network maps each system state to an optimal control action, tuned via cost or reward-based loss functions.
- **Execution:** The trained QFNN outputs the control action through measurement, aiding adaptive scheduling or decision-making.

Here, QFNNs can leverage quantum-enhanced sampling and representation to address combinatorial complexity, potentially outperforming classical methods in structured optimization tasks.

4.8 Practical Insights and Concluding Remarks

- **Depth and Barren Plateaus:** As QFNNs deepen, vanishing gradients may occur, requiring careful initialization or problem-specific circuits.
- **Hardware Constraints:** For NISQ devices, shallow circuits and minimal qubit overhead yield more reliable performance.
- **Data Handling:** Hybrid QFNNs need repeated measurements between layers; coherent QFNNs need more qubits as all remain active.
- **Task-Specific Measurements:** Aligning measurement operators with learning objectives (classification or regression) simplifies training and improves accuracy.

Quantum Feedforward Neural Networks bring classical deep learning paradigms into the quantum domain by replacing linear layers with parameterized quantum gates and using measurement for nonlinearity. They can handle supervised tasks like classification and regression and can be trained with gradient-based or gradient-free optimization in hybrid loops.

While the full potential of QFNNs is still under investigation, their layered structure provides a natural bridge between deep learning and quantum computation, making them a key stepping stone in quantum machine learning.

Quantum Convolutional Neural Networks (QCNNs)