Introduction to quantum computing

Erik Spence

SciNet HPC Consortium

25 July 2022

SciNet compute • calcul CANADA

Today's code and slides

You can get the slides and code for today's class at the SciNet Education web page.

`https://scinet.courses/1231`

Click on the link for the class, and look under "Lectures", click on "Introduction".

About this class, SciNet's SCMP151

The purpose of this course is to introduce you to the basics quantum computing. Some notes about the class:

- We'll meet July 25, 27, 29, from 12:30-3:30pm.
- All classes will be recorded, and the lecture material made available.
- We will be using a workshop format. We will have both lectures and hands-on components.
- Attendance will be taken to determine participation.
- This class qualifies for 9 credits toward a SciNet Scientific Computing Certificate.

Note that this is a SciNet class. It is not an official University of Toronto class. If you ever have questions, please email

courses@scinet.utoronto.ca

About this class, continued

We'll be doing programming of quantum computing circuits in this class. Software you will need:

- I'll be using Python 3.10.X. Python 3.8 or 3.9 will also likely work.
- I won't be teaching Python syntax explicitly, unless asked. Don't be afraid to ask if you see something you don't understand!
- We'll be using the PennyLane quantum computing package. This is installed using pip. There have been some reports of it not installing properly with Python 3.7.

The first class will largely cover the fundamentals. The remaining 2 classes will discuss algorithms and possible applications. Ask questions!

SciNet compute • calcul CANADA

Today's class

Today's class will cover the following topics:

- Introduction to quantum computing, in general.
- Qubits, operators on qubits.
- Quantum circuits.
- PennyLane, programming quantum circuits.
- Multi-qubit systems.
- Entanglement.
- Different hardware approaches.

Please ask questions if something isn't clear.

SciNet compute • calcul CANADA

Quantum computing?

What is quantum computing, as opposed to "classical" computing?

- Regular computers use bits to represent information. These are transistors, or other components, that take on a binary state, a state which represents a value of 1 or 0.
- Quantum computers also use bits to represent information. These are known as "qubits", and can also take a value of 1 or 0.
- These states ("kets") can be thought of as vectors.

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

- Unlike regular bits, qubits can exist in a superposition of states, which is a linear combination of the $|1\rangle$ and $|0\rangle$ states.

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

Where α and β are complex, in general.

SciNet compute • calcul CANADA

Properties of qubits

Qubits form the basis of quantum computing. They are taken to have specific properties.

- All states, $|\mathbf{0}\rangle$ for example, have an associated conjugate transpose vector $\langle \mathbf{0}| = (\mathbf{1} \quad \mathbf{0})$. These are known as "bras".
- All states are taken to be normalized $\langle \boldsymbol{\psi}|\boldsymbol{\psi}\rangle = \mathbf{1}$.
- Because $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$ are orthogonal ($\langle \mathbf{0}|\mathbf{1}\rangle = \mathbf{0}$) they form a basis set. This is known as the "computational basis", and is the most commonly used basis in quantum computing.
- Note that we take the complex conjugate of a state when building the complex transpose

$$\langle \boldsymbol{\psi}|\boldsymbol{\psi}\rangle = \boldsymbol{\alpha}^*\boldsymbol{\alpha} + \boldsymbol{\beta}^*\boldsymbol{\beta} = |\boldsymbol{\alpha}|^2 + |\boldsymbol{\beta}|^2 = \mathbf{1}$$

- The $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ values correspond to the probability amplitudes of $|\boldsymbol{\psi}\rangle$ being measured in either state $|\mathbf{0}\rangle$ or $|\mathbf{1}\rangle$.

The power of a quantum computer is in the fact that a quantum state is a superposition of its basis states.

Measurement

Because we are dealing with quantum states we are naturally dealing with probabilities.

- The final operation of a quantum circuit is a measurement of some or all of the qubits.
- However, we will measure qubits to be in specific states with only a certain probability, as given by the probability amplitudes of the states.
- As such, we must rerun the calculation many times to determine the probabilities associated with each state.

If $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ then the probability of measuring

- the state $|0\rangle$ is $|\alpha|^2$ and
- the state $|1\rangle$ is $|\beta|^2$.

The probabilities are often represented as $\langle 0|\psi\rangle$, since they correspond to the magnitude of the projection of the state $|\psi\rangle$ onto the basis $\langle 0|$.

Operating on quantum states

Having quantum states is great, but we need to be able to manipulate them.

- Ideally, we would like to be able to modify them as we wish:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \rightarrow |\psi'\rangle = \alpha'\,|0\rangle + \beta'\,|1\rangle$$

- Since our basis states have 2 elements we need a 2 x 2 matrix to perform this operation.
- This matrix must conserve the normalization of the final state $|\psi'\rangle$.
- The class of matrices that satisfy these conditions are the Unitary Matrices: $\boldsymbol{U}\boldsymbol{U}^\dagger = \boldsymbol{I}$.
- $\boldsymbol{U}$ is the unitary matrix, $\boldsymbol{U}^\dagger$ is its conjugate transpose, and $\boldsymbol{I}$ is the identity matrix.

Now that we know how to operate on qubits we're ready to do more interesting things.

Important operators

There are several quantum operators that are commonly used.

- Hadamard gate:

$$\boldsymbol{H}|\mathbf{0}\rangle = |+\rangle = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\left(|\mathbf{0}\rangle + |\mathbf{1}\rangle\right), \qquad \boldsymbol{H}|\mathbf{1}\rangle = |-\rangle = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\left(|\mathbf{0}\rangle - |\mathbf{1}\rangle\right)$$

$$\boldsymbol{H} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\begin{bmatrix}\mathbf{1} & \mathbf{1}\\ \mathbf{1} & \mathbf{-1}\end{bmatrix}$$

This gate creates a uniform superposition of the basis states $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$. These states are so common that they get their own symbols, $|+\rangle$ and $|-\rangle$. The Hadamard operator is commonly used at the beginning of quantum circuits, as we'll see later.

Note further that it is its own inverse: $\boldsymbol{H^2} = \boldsymbol{I}$.

Important operators, continued

More commonly encountered quantum operators include:

- Pauli X gate (also known as the "bit flip" or "NOT gate"):

$$\boldsymbol{X} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{bmatrix}, \qquad \boldsymbol{X}\left|\mathbf{0}\right\rangle = \left|\mathbf{1}\right\rangle, \qquad \boldsymbol{X}\left|\mathbf{1}\right\rangle = \left|\mathbf{0}\right\rangle$$

- RZ gate: if $\left|\psi\right\rangle = \alpha\left|0\right\rangle + \beta\left|1\right\rangle$, then

$$\boldsymbol{RZ}(\omega) = \begin{bmatrix} e^{-i\frac{\omega}{2}} & \mathbf{0} \\ \mathbf{0} & e^{i\frac{\omega}{2}} \end{bmatrix} \qquad \boldsymbol{RZ}(\omega)\left|\psi\right\rangle = \alpha\left|0\right\rangle + \beta e^{i\omega}\left|1\right\rangle$$

This gate changes the relative phase between $\left|0\right\rangle$ and $\left|1\right\rangle$. Note that global phases do not affect the probability of measuring the state in question, and are thus usually ignored.

Important operators, continued more

More common quantum operators:

- Pauli Z gate (equivalent to $RZ(\pi)$):

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad Z|0\rangle = |0\rangle, \qquad Z|1\rangle = -|1\rangle$$

.

This gate is its own inverse. $Z^2 = I$.

- S gate (equivalent to $RZ(\pi/2)$):

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad S|0\rangle = |0\rangle, \qquad S|1\rangle = i|1\rangle$$

.

$S^2 = Z$.

Important operators, continued even more

A few other quantum operators you may run into:

- $\boldsymbol{RX(\theta)}$ gate:

$$\boldsymbol{RX(\theta)} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

- $\boldsymbol{RY(\theta)}$ gate:

$$\boldsymbol{RY(\theta)} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ -\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

These represent rotations about the $\boldsymbol{X}$ and $\boldsymbol{Y}$ axes, if you represent the state a vector in 3-space.

The goal

What can we do with all we've got so far? What are we trying to accomplish?

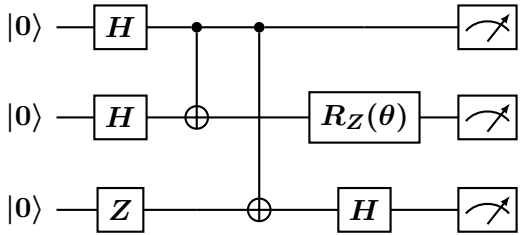
- We want to manipulate quantum states using our operators.
- These quantum states will represent some physical or numerical quantity.
- The operations on the states will be chosen so as to implement the algorithm that is desired.
- Such a set of operations and states is known as a "quantum circuit".
- Once we've done all of the operations that we need, we will measure the final quantum states.
- These measurements will be used to give the answer to whatever calculation we are after.

This is the end point of this whole line of work. Of course, crafting algorithms that can accomplish specific tasks can be non-trivial, and is an active area of research.

SciNet compute • calcul CANADA

Quantum circuits

Quantum circuits are organized by qubit, as per the image below.



Time runs from left to right. Each "wire" in the circuit represents a qubit. The various gates and operations are represented by boxes or connections between the wires. The rightmost symbol is a measurement.

PennyLane

It's time to start coding. We will be using the PennyLane quantum computing framework.

- Developed by Xanadu (`https://xanadu.ai`).
- Open source, with a Python interface.
- Specifically designed for differentiable programming of quantum computers.
- Implemented as both a quantum circuit simulator and a front end to a multitude of different quantum-computer back ends.
- In today's class we will be only using the quantum simulator. Getting access to Xanadu's cloud-accessible 7-qubit system is an option if you sign up.

PennyLane can be installed into your Python virtual environment in the usual way.

```
ejspence@mycomp ~>
ejspence@mycomp ~> pip install pennylane
ejspence@mycomp ~>
```

Quantum computing frameworks, an aside

PennyLane is not the only quantum-computing framework available. There are many other options out there.

- Cirq, Google, released 2018.
- Qiskit (Quantum Information Science Kit), IBM, 2017.
- Forest, Rigetti, 2017. This actually refers to the interface to Rigetti's pyQuil package.
- Quantum Development Kit (QDK), Microsoft, based on its own language "Q#", 2018.
- And many many others.

A more comprehensive list can be found here:
`https://quantiki.org/wiki/list-qc-simulators`.

SciNet compute • calcul CANADA

PennyLane, continued

In PennyLane computations are represented as "quantum node" objects. These are used to define the circuit and to specify the device that will execute the calculation.

Here we use the "default.qubit" device, which is a standard quantum simulator.

The number of "wires" indicates the number of qubits in our circuit.

The circuit itself is represented as a function. By default all qubits are initialized as $|\mathbf{0}\rangle$.

```
In [1]: import numpy as np
In [2]: import pennylane as qml
In [3]:
In [3]: dev = qml.device('default.qubit',
...:            wires = 1)
In [4]:
In [4]: def q_circuit():
...:        qml.Hadamard(wires = 0)
...:        qml.RY(np.pi/3, wires = 0)
...:        return qml.probs(wires = 0)
In [5]:
In [5]: qnode = qml.QNode(q_circuit, dev)
In [6]:
In [6]: qnode()
Out[6]: tensor([0.0669873, 0.9330127],
                requires_grad=True)
```

PennyLane, continued more

Rather than build our "quantum node" explicitly, it's easier to build it using a Python decorator.

If you're not familiar with Python decorators, they are invoked using the "@" call before the function definition.

Many things can be returned by the circuit. Here we return the final quantum state.

```
In [7]:
In [7]: dev = qml.device('default.qubit',
...:            wires = 1)
In [8]:
In [8]: @qml.qnode(dev)
...:    def q_circuit2():
...:        qml.Hadamard(wires = 0)
...:        return qml.state()
In [9]:
In [9]: q_circuit2()
Out[9]: tensor([0.70710678+0.j, 0.70710678+0.j],
                requires_grad=True)
In [10]:
```

Hands-on 1

Now that we know how to program basic quantum circuits, it's time to start playing with PennyLane. See if you can create single-wire circuit functions that return the following quantum states:

1. $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$,
2. $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}e^{\frac{3}{4}i\pi}|1\rangle$, up to a global phase
3. $-i|1\rangle$
4. $-i|0\rangle$
5. $\frac{\sqrt{3}}{2}|0\rangle - \frac{i}{2}|1\rangle$

The PennyLane functions PauliX, Hadamard, RZ and RX will be useful.

Multi-qubit systems

Thus far we've only been looking at single-qubit quantum states.

- These states can exist as a superposition of the basis states.
- Thus far, the only basis states we've been considering are the "computational" basis states, $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$.
- Other basis states are possible, and are sometimes used, such as $|+\rangle$ and $|-\rangle$.
- When we deal with multi-qubit states, we deal with a superposition of all the underlying qubits which are involved.
- This superposition is built using a tensor product.

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}$$

.

Multi-qubit systems, continued

Our computational basis set is now represented by the tensor product of the two single qubits.

$$|0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

These states are represented with the symbols $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ respectively.

Multi-qubit systems in PennyLane

Initializing a multi-qubit system in PennyLane is as simple as specifying more than one wire.

Note how the state of the system is represented by a single 4-element vector, and the state is initialized to $|\mathbf{00}\rangle$.

```
In [10]:
In [10]: dev = qml.device('default.qubit',
...:                      wires = 2)
In [11]:
In [11]: @qml.qnode(dev)
...:     def my_mqubit():
...:         return qml.state()
In [12]:
In [12]: my_mqubit()
Out[12]: tensor([1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
                requires_grad=True)
In [13]:
```

Multi-qubit operations

Operations on multiple qubits are expressed as the tensor product of the operators involved.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} \alpha & \beta \\ \delta & \gamma \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} \alpha & \beta \\ \delta & \gamma \end{bmatrix} & b\begin{bmatrix} \alpha & \beta \\ \delta & \gamma \end{bmatrix} \\ c\begin{bmatrix} \alpha & \beta \\ \delta & \gamma \end{bmatrix} & d\begin{bmatrix} \alpha & \beta \\ \delta & \gamma \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a\alpha & a\beta & b\alpha & b\beta \\ a\delta & a\gamma & b\delta & b\gamma \\ c\alpha & c\beta & d\alpha & d\beta \\ c\delta & c\gamma & d\delta & d\gamma \end{bmatrix}$$

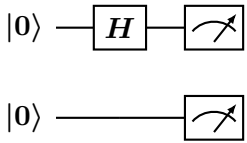
.

The Unitary nature of the original operators means that the combined operator will also be Unitary.

Multi-qubit operations, example 1

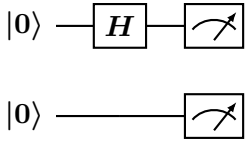
Consider this situation:

- We have a 2-qubit state.
- We want to perform the Hadamard operator, but only on the first qubit.
- What is the resulting state?



This operation corresponds to the above circuit.

Multi-qubit operations, example 1, continued



$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\otimes\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}$$

.

The second operator is just the identity matrix, which is, of course, also Unitary.

Multi-qubit operations, example 1, continued more

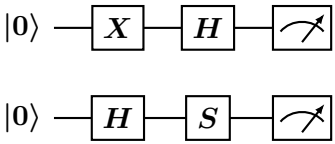
$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\1\\0\end{bmatrix}$$

.

Note that the starting state is $|\mathbf{0}\rangle \otimes |\mathbf{0}\rangle = |\mathbf{00}\rangle$, and the final state is $\frac{1}{\sqrt{2}}(|\mathbf{00}\rangle + |\mathbf{10}\rangle)$.

```
In [13]:
In [13]: dev = qml.device('default.qubit',
...:                      wires = 2)
In [14]:
In [14]: @qml.qnode(dev)
...:     def my_mqubit2():
...:         qml.Hadamard(wires = 0)
...:         return qml.state()
In [15]:
In [15]: my_mqubit2()
Out[15]: tensor([0.70710678+0.j, 0.  +0.j,
                 0.70710678+0.j, 0.  +0.j],
                 requires_grad=True)
In [16]:
```

Multi-qubit operations, example 2



$$X \otimes H = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

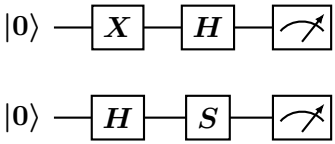
$$H \otimes S = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & i & 0 & i \\ 1 & 0 & -1 & 0 \\ 0 & i & 0 & -i \end{bmatrix}$$

Multi-qubit operations, example 2, continued

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & i & 0 & i\\ 1 & 0 & -1 & 0\\ 0 & i & 0 & -i\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}0 & 0 & 1 & 1\\ 0 & 0 & 1 & -1\\ 1 & 1 & 0 & 0\\ 1 & -1 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & i & 0 & i\\ 1 & 0 & -1 & 0\\ 0 & i & 0 & -i\end{bmatrix}\begin{bmatrix}0\\0\\1\\1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}1\\i\\-1\\-i\end{bmatrix}$$

Note that, as the circuit is read left-to-right, the orientation of the matrix operators, relative to the initial state vector, must be reversed.

Multi-qubit operations, example 2, continued more



$$\frac{1}{2}\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & i & 0 & i \\ 1 & 0 & -1 & 0 \\ 0 & i & 0 & -i \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ i \\ -1 \\ -i \end{bmatrix}$$

```
In [16]: dev = qml.device('default.qubit',
...:                      wires = 2)
In [17]:
In [17]: @qml.qnode(dev)
...:     def my_mqubit3():
...:         qml.PauliX(wires = 0)
...:         qml.Hadamard(wires = 1)
...:         qml.Hadamard(wires = 0)
...:         qml.S(wires = 1)
...:         return qml.state()
In [18]:
In [18]: my_mqubit3()
Out[18]: tensor([0.5 +0.j, 0.  +0.5j,
                 -0.5+0.j, -0.  -0.5j],
                 requires_grad=True)
In [19]:
```

Entanglement

Superposition is the first feature of quantum computing that distinguishes it from classical computing. The second property which distinguishes quantum computing is entanglement.

Suppose we have two single-qubit states.

$$|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle, \qquad |\psi_2\rangle = \gamma|0\rangle + \delta|1\rangle$$

What is the tensor product of these two states?

$$\begin{aligned} |\psi_1\rangle \otimes |\psi_2\rangle &= (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) \\ &= \alpha|0\rangle \otimes (\gamma|0\rangle + \delta|1\rangle) + \beta|1\rangle \otimes (\gamma|0\rangle + \delta|1\rangle) \\ &= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \end{aligned}$$

Entanglement, continued

Two single-qubit states combine thus:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \alpha\gamma\,|00\rangle + \alpha\delta\,|01\rangle + \beta\gamma\,|10\rangle + \beta\delta\,|11\rangle$$

Now consider the state

$$|\phi\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

As you can see, there is no way to express the state $|\phi\rangle$ as a combination of two single-qubit states. An entangled state is a state that cannot be described as the tensor product of two other states. As such, it can only be described by writing out the full state.

States that can be described in terms of smaller states are called "separable".

Creating entangled states

How does one create an entangled state?

- Thus far all multi-qubit operations that we've performed involved tensor products of single-qubit operations.
- To create an entangled state we need an "entangling gate".
- These gates transform a separable state into an entangled state.
- Like entangled states, entangled gates cannot be written as a tensor product of smaller operations.

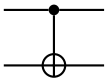
Entangling gates form an important part of quantum computing.

SciNet compute • calcul CANADA

Controlled-NOT gate

The CNOT (controlled-NOT gate) is an important entangled gate.

- A two-qubit gate,
- the first qubit is the solid dot, the "control qubit",
- the second qubit is the circle-cross symbol, the "target qubit",
- if the control qubit is "1", the Pauli X operation is performed on the target qubit.



$$\mathbf{CNOT} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix}$$

This representation of $\mathbf{CNOT}$ assumes that the control bit is the first wire, and the target bit the second.

Controlled-NOT gate, continued

$$\mathbf{CNOT}\,|00\rangle = |00\rangle$$
$$\mathbf{CNOT}\,|01\rangle = |01\rangle$$
$$\mathbf{CNOT}\,|10\rangle = |11\rangle$$
$$\mathbf{CNOT}\,|11\rangle = |10\rangle$$

Again, this assumes that the control bit is the first bit.

The first wire indicated in the code is the control bit, the second is the target.

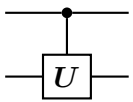
```
In [19]: dev = qml.device('default.qubit',
...:                      wires = 2)
In [20]:
In [20]: @qml.qnode(dev)
...:     def my_cnot(input_wires):
...:         qml.PauliX(wires = 0)
...:         qml.CNOT(wires = input_wires)
...:         return qml.state()
In [21]:
In [21]: my_cnot([0, 1])
Out[21]: tensor([0.+0.j, 0.+0.j, 0.+0.j, 1.+0j],
                requires_grad=True)
In [22]:
In [22]: my_cnot([1, 0])
Out[22]: tensor([0.+0.j, 0.+0.j, 1.+0.j, 0.+0j],
                requires_grad=True)
```

Universal controlled gates

The CNOT is not the only operation to which we can apply control.

In general we can apply control to any Unitary operation.

Once again, this representation assumes that the control bit is the first wire, and the target bit the second.



$$\mathbf{CU} = \begin{bmatrix} \boldsymbol{I_2} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{U} \end{bmatrix}$$

Where $\boldsymbol{I_2}$ is the 2 x 2 Identity matrix, and $\boldsymbol{U}$ is the Unitary operator we are controlling.

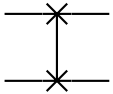
Other controlled gates

The Controlled-Z gate is another you may encounter.

$$\mathbf{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Also known as the CZ gate, or the controlled phase gate.

The SWAP gate does exactly that: it swaps the states of the 2 qubits.



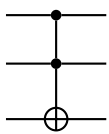
$$\mathbf{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Other controlled gates, continued

We aren't restricted to 2-qubit gates.

Another important gate is the Toffoli gate, which is essentially a controlled-CNOT gate. It has 2 control qubits and a single target qubit.

This is by-far the most common larger gate.



$$\mathbf{TOF} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix}$$

SciNet compute • calcul CANADA

Implementing quantum computers

Now that you have a sense of how a quantum calculation might be performed, you can start to see some of the characteristics we would expect a quantum computer to have.

- We need qubits! These must be able to be in a superposition of computational basis states.
- The more qubits we can have interacting, the better.
- We need qubits that can entangle with each other.
- We need to be able to initialize the qubits into a specific state.
- We need the qubits to hold their states (this is a big problem!).
- We need to be able to apply operations to the qubits.
- We need to be able to measure the qubits.

What sort of hardware might work for this?

SciNet compute • calcul CANADA

Types of quantum computers

What sort of hardware is used for these (gate-based) quantum computers?

- Neutral atom: magneto-optical traps are used to trap Cesium or Rubidium. These are cooled to mK temperatures, and put into an array, and manipulated.
- Nitrogen-vacancy center-in-diamond: a diamond is built with a pair of missing carbon atoms, which are replaced with nitrogen. The resulting defect is manipulated using microwaves.
- Photonics: optical techniques are used to create qubits. However, photons cannot interact in a vacuum, but only indirectly through another medium.
- Spin-Qubits: silicon-based quantum dots.
- Superconducting qubits: a Cooper pair is joined to a Josephson junction. Unitary operations are applied using microwaves.
- Trapped Ion.
- And many others.

SciNet compute • calcul CANADA

Summary

A summary of today's class.

- Quantum computers operate on qubits, rather than regular bits.
- These qubits can be in a superposition of states.
- Operations are performed on qubits to manipulate their values, and how they relate to each other.
- Algorithms which manipulate qubit are expressed as quantum circuits.
- Multi-qubit systems involve superpositions of multiple qubits. These are built by taking the tensor product of the underlying qubits.
- An entangled state is a quantum state that cannot be expressed as a tensor product of smaller quantum states.
- Entangled states are built using entangling operators.

We will start building quantum algorithms next class.

SciNet compute • calcul CANADA

Hands-on 2

Let's play with some multi-qubit circuits.

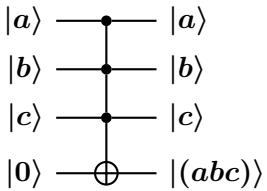
1. See if you can create two-qubit circuit functions that return the following quantum states (these are known as the Bell states):
   1. $|\psi_+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$,
   2. $|\psi_-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$,
   3. $|\phi_+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$,
   4. $|\phi_-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$.
2. Write a function that implements the SWAP gate, using only CNOT gates. How many CNOT gates are necessary?

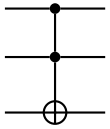
The PennyLane functions PauliX, Hadamard, CNOT and PauliZ will be useful.

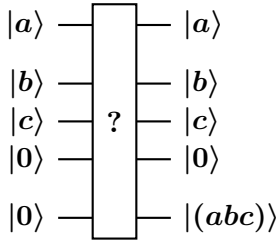
SciNet compute • calcul CANADA

Hands-on 2, continued

3. Consider the controlled-controlled-CNOT gate (left). Implement this gate using only the Toffoli gate (centre), and an extra auxiliary qubit (right).







The auxiliary qubit should both start and end in the $|0\rangle$ state. The PennyLane Toffoli function takes a list of 3 wires as its argument. The first two wires are the control, the third is the target.

Linky goodness

PennyLane:

- https://github.com/PennyLaneAI/pennylane
- https://www.qmunity.tech/tutorials/an-introduction-to-pennylane

Quantum computing:

- https://xanadu.ai
- https://codebook.xanadu.ai
- https://www.scientificamerican.com/video/how-does-a-quantum-computer-work
- https://www.epiqc.cs.uchicago.edu/qc-introduction

SciNet compute • calcul CANADA