CPEN 400Q Lecture 07
Expectation values; introducing the variational quantum classifier

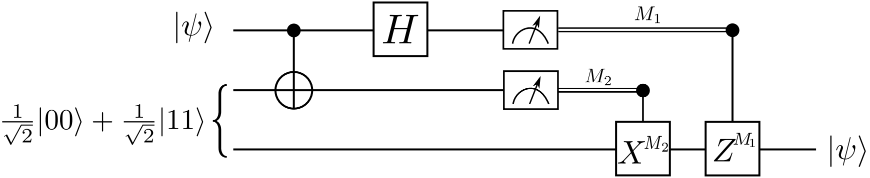
Monday 30 January 2023

Announcements

- Quiz 3 today at beginning of class
- Assignment 1 available

Last time

We teleported a qubit!



Learning outcomes

- Compute expectation values of observables
- Describe the main structural elements of a variational quantum algorithm
- Find optimal parameters of a variational circuit in PennyLane

Generally, we are interested in measuring real, physical quantities. In physics, these are called observables.

Observables are represented mathematically by Hermitian matrices. An operator (matrix) H is Hermitian if

$$H = H^{\dagger}$$

Why Hermitian? The possible measurement outcomes are given by the eigenvalues of the operator, and eigenvalues of Hermitian operators are **real**.

Example:

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

Z is Hermitian:

Its eigensystem is

Example:

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

X is Hermitian and its (normalized) eigensystem is

Example:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Y is Hermitian and its (normalized) eigensystem is

Expectation values

When we measure X, Y, or Z on a state, for each shot we will get one of the eigenstates (/eigenvalues).

If we take multiple shots, what do we expect to see *on average*?

Analytically, the **expectation value** of measuring the observable M given the state $|\psi\rangle$ is

Expectation values: analytical

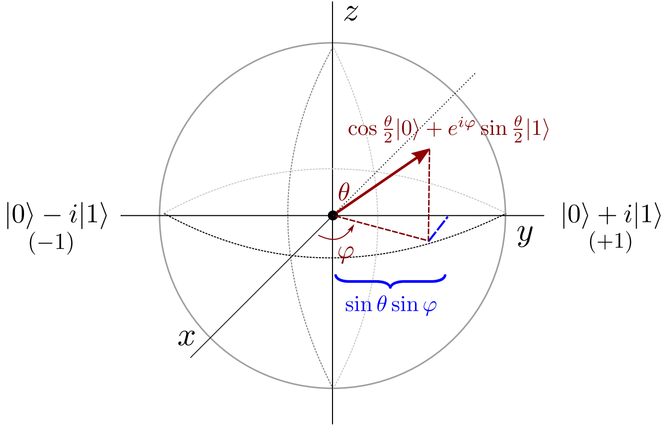
Exercise: consider the quantum state

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

Compute the expectation value of Y:

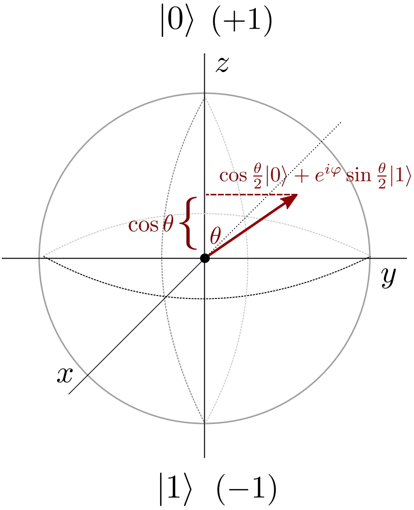
Expectation values and the Bloch sphere

The Bloch sphere offers us some more insight into what a projective measurement is.



Exercise: derive the expression in blue by computing $\langle\psi|\, Y |\psi\rangle$.

Expectation values and the Bloch sphere



Expectation values: from measurement data

Let's compute the expectation value of Z for the following circuit using 10 samples:

```
dev = qml.device('default.qubit', wires=1, shots=10)

@qml.qnode(dev)
def circuit():
    qml.RX(2*np.pi/3, wires=0)
    return qml.sample()
```

Results might look something like this:

$$[1,\ 1,\ 1,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1]$$

The expectation value pertains to the measured eigenvalue; recall Z eigenstates are

$$\lambda_1 = +1, \qquad |\psi_1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\lambda_2 = -1, \qquad |\psi_2\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

So when we observe $|0\rangle$, this is eigenvalue $+1$ (and if $|1\rangle$, -1). Our samples shift from

```
[1, 1, 1, 0, 1, 1, 1, 0, 1, 1]
```

to

```
[-1, -1, -1, 1, -1, -1, -1, 1, -1, -1]
```

Expectation values: from measurement data

The expectation value is the weighted average of this, where the weights are the eigenvalues:

where

- n_1 is the number of $+1$ eigenvalues
- n_{-1} is the number of -1 eigenvalues
- N is the total number of shots

For our example,

Let's do this in PennyLane instead:

```
dev = qml.device('default.qubit', wires=1)

@qml.qnode(dev)
def measure_z():
    qml.RX(2*np.pi/3, wires=0)
    return qml.expval(qml.PauliZ(0))
```

Multi-qubit expectation values

Example: operator $Z \otimes Z$.

Eigenvalues are computational basis states:

$$
\begin{aligned}
(Z \otimes Z)|00\rangle &= |00\rangle \\
(Z \otimes Z)|01\rangle &= -|01\rangle \\
(Z \otimes Z)|10\rangle &= -|10\rangle \\
(Z \otimes Z)|11\rangle &= |11\rangle
\end{aligned}
$$

To compute an expectation value from data:

$$\langle Z \otimes Z \rangle = \frac{1 \cdot n_1 + (-1) \cdot n_{-1}}{N}$$

Example: operator $X \otimes I$.

Eigenvalues of X are the $|+\rangle$ and $|-\rangle$ states:

$$
\begin{aligned}
(X \otimes I)|+0\rangle &= |+0\rangle \\
(X \otimes I)|+1\rangle &= |+1\rangle \\
(X \otimes I)|-0\rangle &= -|-0\rangle \\
(X \otimes I)|-1\rangle &= -|-1\rangle
\end{aligned}
$$

Fun fact: All Pauli operators have an equal number of $+1$ *and* -1 *eigenvalues!*

How to compute expectation value of X from data, when we can only measure in the computational basis?

Basis rotation: apply H to first qubit

$$
\begin{aligned}
(H \otimes I)(X \otimes I)|+0\rangle &= |00\rangle \\
(H \otimes I)(X \otimes I)|+1\rangle &= |01\rangle \\
(H \otimes I)(X \otimes I)|-0\rangle &= -|10\rangle \\
(H \otimes I)(X \otimes I)|-1\rangle &= -|11\rangle
\end{aligned}
$$

When we measure and obtain $|10\rangle$ or $|11\rangle$, we know those correspond to the -1 eigenstates of $X \otimes I$.

Multi-qubit expectation values can be created using the `@` symbol:

```
@qml.qnode(dev)
def circuit(x):
    qml.Hadamard(wires=0)
    qml.CRX(x, wires=[0, 1])
    return qml.expval(qml.PauliZ(0) @ qml.PauliZ(1))
```

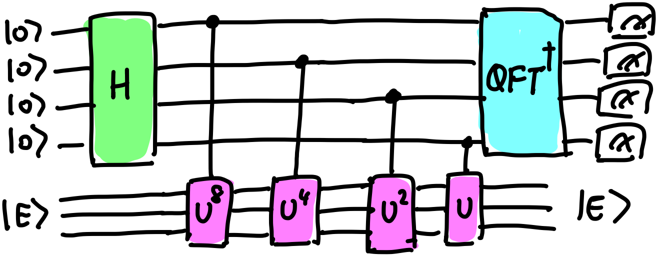
Hands-on: multi-qubit expectation values

Can also return *multiple* expectation values, if there are no shared qubits.

```
@qml.qnode(dev)
def circuit(x):
    qml.Hadamard(wires=0)
    qml.CRX(x, wires=[0, 1])
    return qml.expval(qml.PauliZ(0)), qml.expval(qml.
                                    PauliZ(1))
```

The quantum algorithms of tomorrow will be *big*.

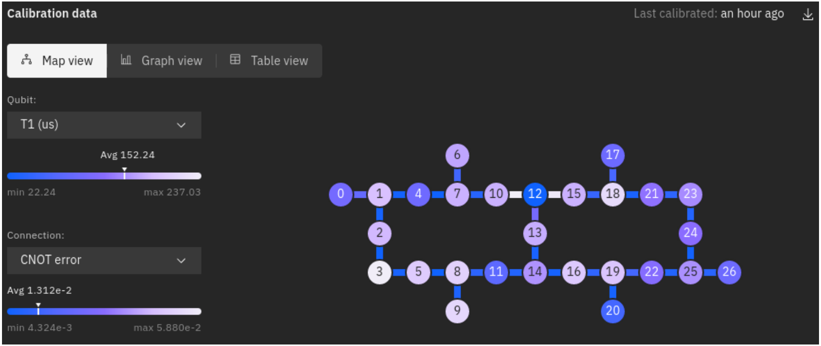
- Use many qubits
- Require dense qubit *connectivity*
- High circuit depth



(This is phase estimation: we will see it in a few weeks.)

Why variational algorithms?

Today's quantum computers today aren't really suitable for these...



[A NISQ-era device, for exemplary purposes]

Image credit: IBM Q Auckland, screen capture 2022-03-01.

`https://quantum-computing.ibm.com/services?services=systems&system=ibm_auckland`

Why variational algorithms?

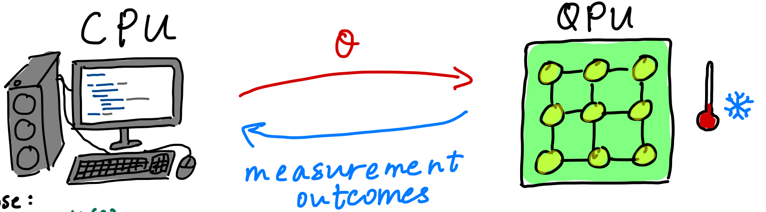
What *can* we do with a NISQ device?

Suitable algorithms should:

- Not be too long
- Fit the processor architecture well
- Use a quantum computer to do something non-trivial
- Still solve an interesting problem

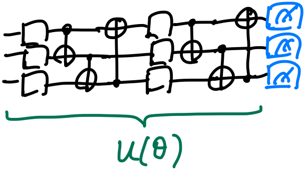
Variational algorithms

Feature an iterative exchange between classical and quantum devices. (Sometimes called "hybrid" quantum-classical algorithms)



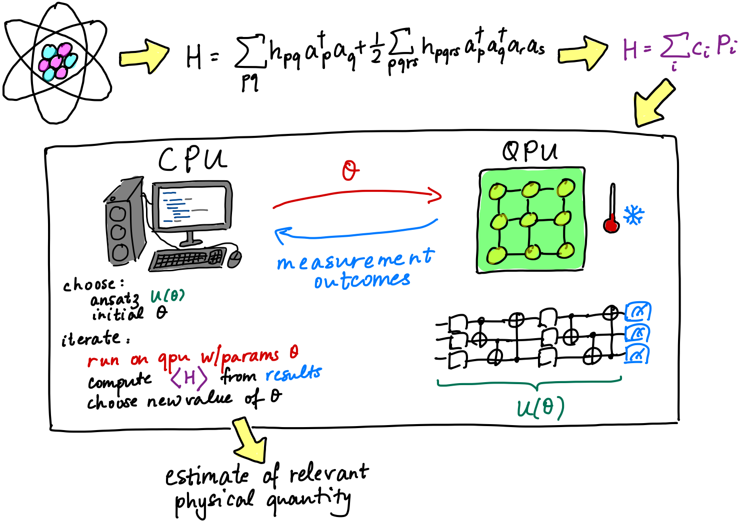
choose:
- ansatz $U(\theta)$
- initial θ

iterate:
- run on qpu w/params θ
- compute cost from results
- choose new value of θ



Variational algorithms

Useful in many domains: quantum chemistry, quantum machine learning, optimization, etc.



Parametrized quantum circuits

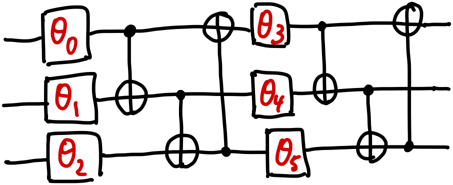
Some quantum operations depend on real-valued *parameters*.

These can be passed as arguments to a quantum function.

```
def circuit(x, y, z):
    qml.RX(x, wires=0)
    qml.RY(y, wires=1)
    qml.RZ(z, wires=2)
```

We call these **parametrized quantum circuits**.

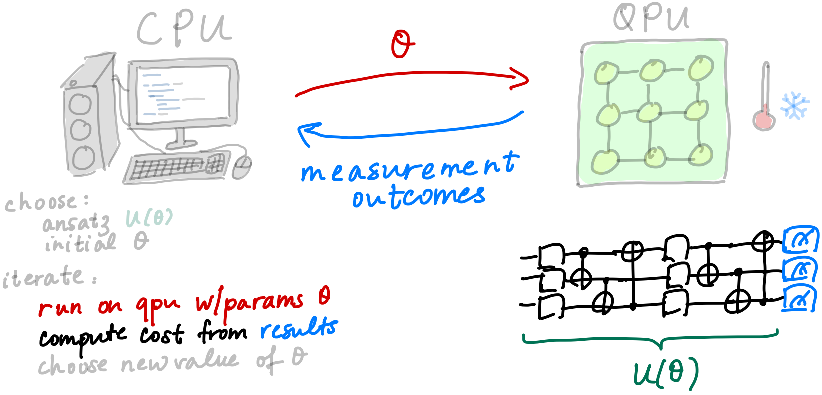
Parametrized quantum circuits




```
def parametrized_circuit(theta):
    qml.RX(theta[0], wires=0)
    qml.RX(theta[1], wires=1)
    qml.RX(theta[2], wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
    qml.RX(theta[3], wires=0)
    qml.RX(theta[4], wires=1)
    qml.RX(theta[5], wires=2)
    qml.CNOT(wires=[0, 1])
    qml.CNOT(wires=[1, 2])
    qml.CNOT(wires=[2, 0])
```

Parametrized quantum circuits

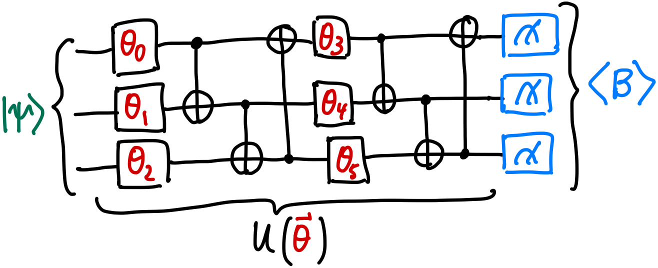
Parametrized circuits are used to assist in evaluation of a cost function which represents a particular problem.



Parametrized quantum circuits

We are trying to *find optimal values* for these parameters in order to minimize the cost, which represents the solution to the problem.

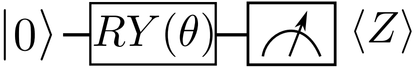
Expectation values are often used to construct a cost function.



$$\min_{\vec{\theta}} \langle B \rangle = \min_{\vec{\theta}} \langle \psi | U^\dagger(\vec{\theta}) \, B \, U(\vec{\theta}) | \psi \rangle$$

Expectation values and objective functions

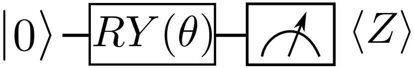
Example: find the value of θ which minimizes $\langle Z \rangle$.



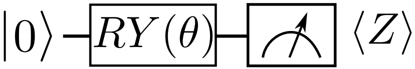
Key point: the expectation values measured at the end are *functions* of the variational parameters, i.e.,

We can compute such functions, then differentiate them.

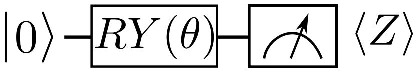
Expectation values and objective functions



Let's compute the analytical expression for $\langle Z\rangle$:



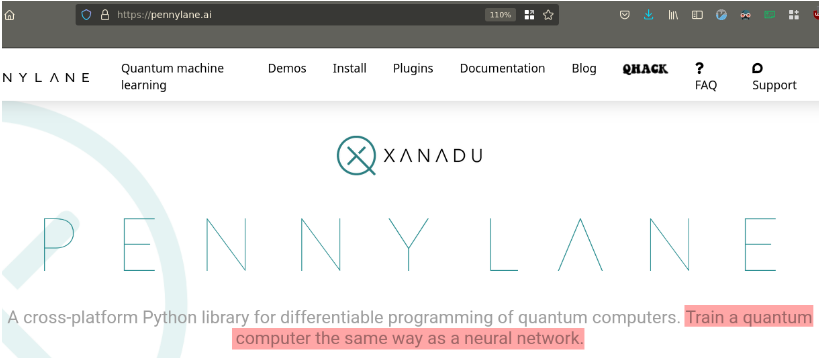
Let's compute the analytical expression for $\langle Z \rangle$:



Compute the derivative:

But obviously, we don't want to do this by hand... PennyLane will do it for us!

Training variational quantum circuits



Training variational quantum circuits

Circuits can be trained using standard optimization techniques *on a classical computer* such as gradient descent.

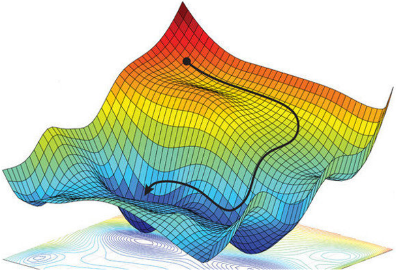


Image credit: A. Amini, A. P. Soleimany, S. Karaman, D. Rus. *Spatial Uncertainty Sampling for End-to-End Control*. NIPS 2017.

`qml.grad`

`qml.grad` is a *transform*: apply to a QNode to obtain a function that computes the *gradient* of that QNode.

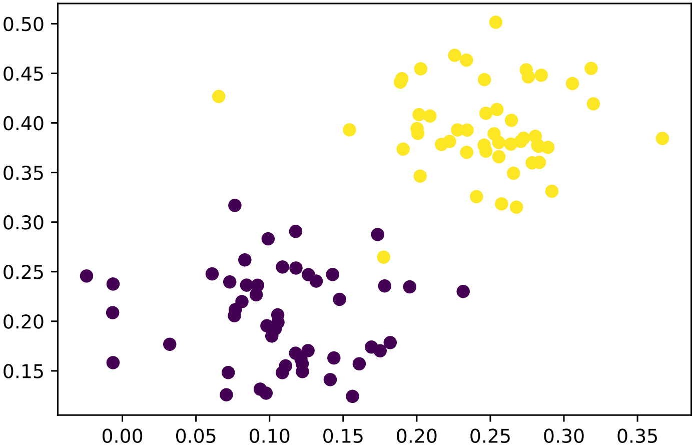
```
@qml.qnode(dev)
def pqc(theta):
    qml.RY(theta, wires=0)
    return qml.expval(qml.PauliZ(0))

grad_fn = qml.grad(pqc)
grad_fn(theta)
```

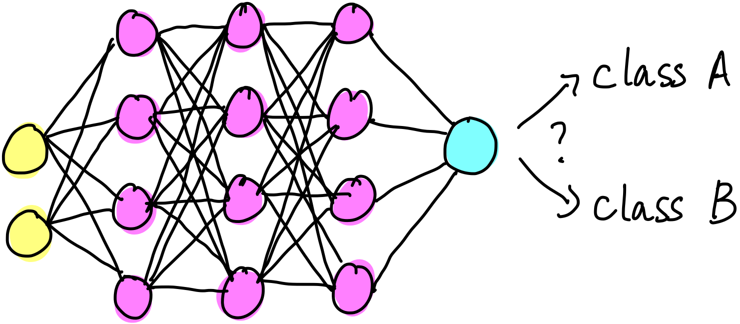
Later in the course, we will learn about how the gradient is actually evaluated on a quantum computer.

Overarching problem: binary classification

Suppose we have some 2-dimensional data:



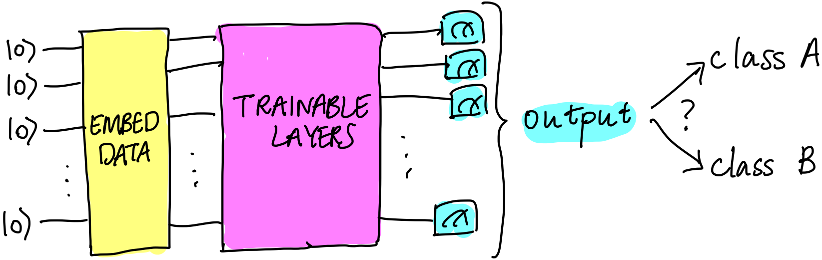
Consider how classification can be done with a neural network:



Overarching problem: binary classification

We are going to train a quantum circuit to *classify* this data.

The general structure of our model is:



Recap

- Compute expectation values of observables
- Describe the main structural elements of a variational quantum algorithm
- Find optimal parameters of a variational circuit in PennyLane

Next time

Content:

- Classifying data with the VQC

Action items:

1. Assignment 1 (can do all problems now)

Recommended reading:

- QML glossary entries (`https://pennylane.ai/qml/glossary.html`):
  - Quantum differentiable programming
  - Quantum gradients
  - Variational circuit
- `https://arxiv.org/abs/2012.09265v2` (review paper)