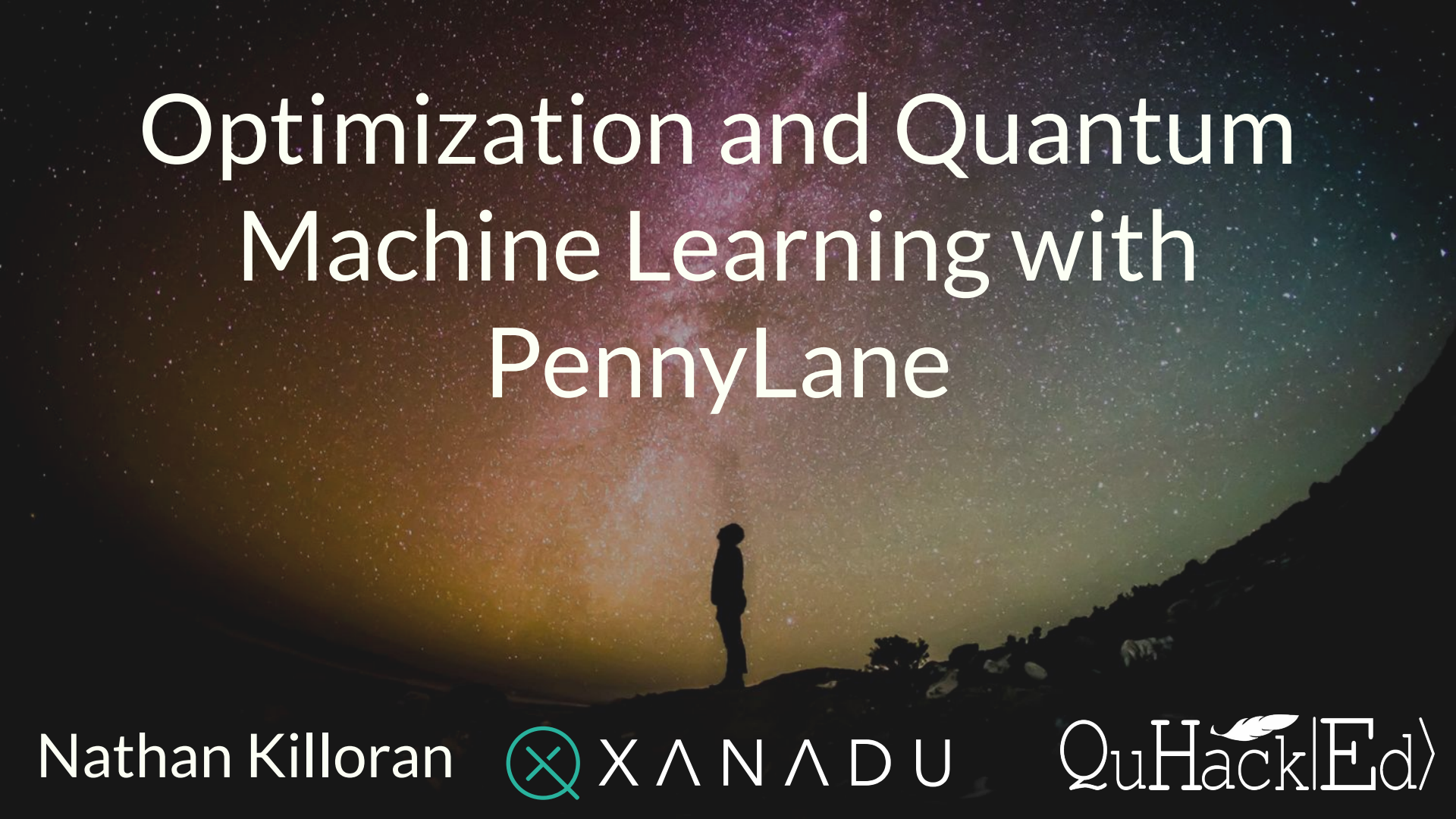
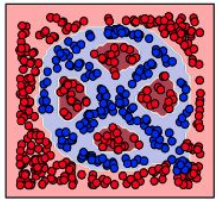
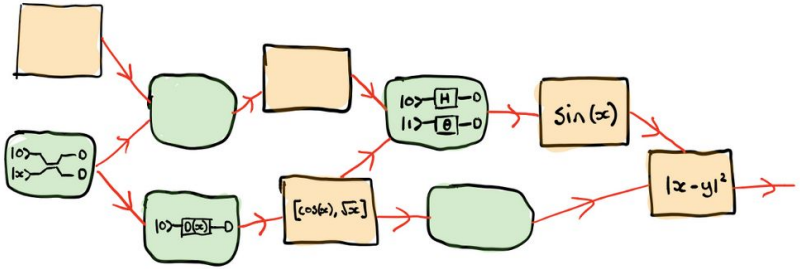

Optimization and Quantum Machine Learning with PennyLane
Nathan Killoran
XANADU
QuHack|Ed⟩

Outline

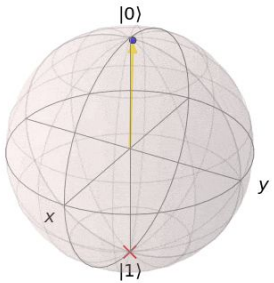
1. Quantum machine learning



2. Training quantum circuits



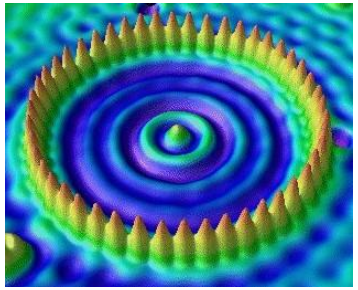
3. PennyLane + examples



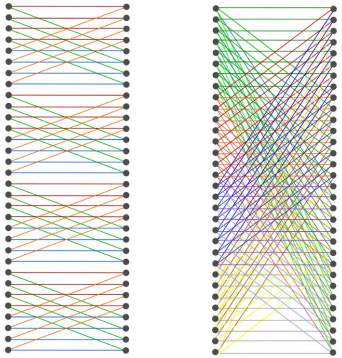
QUANTUM MACHINE LEARNING

Quantum computers are good at:

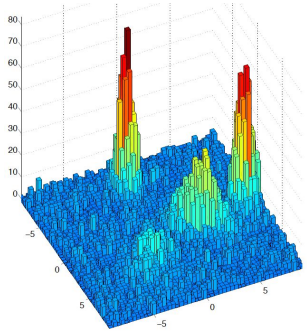
Quantum physics



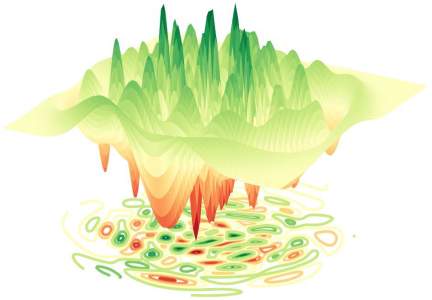
Linear algebra



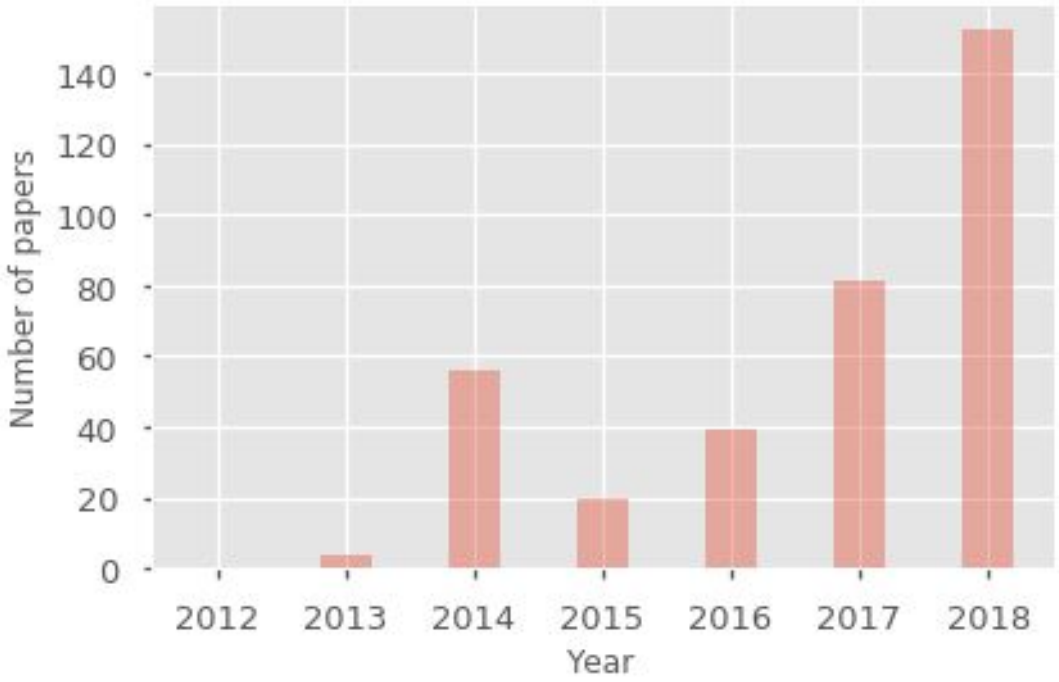
Sampling



Optimization

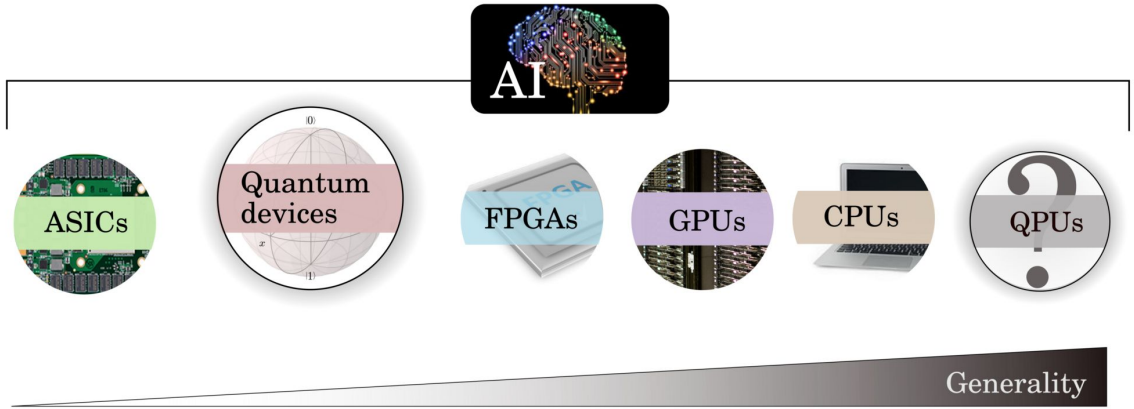


Quantum Machine Learning papers



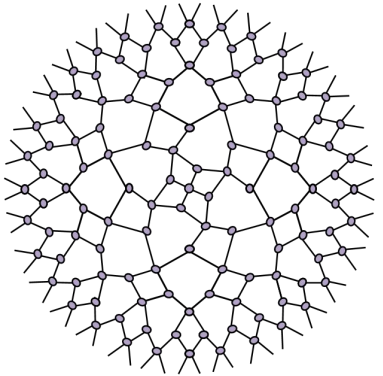
Quantum Machine Learning

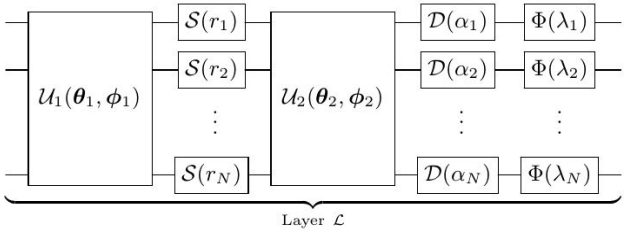
- AI/ML already uses special-purpose processors: GPUs, TPUs, ASICs
- Quantum computers (QPUs) could be used as special-purpose AI accelerators
- May enable training of previously intractable models

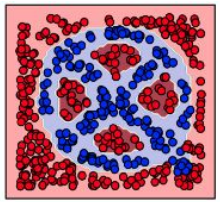


New AI models

- Quantum computing can also lead to new machine learning models
- Examples currently being studied are:
  - Kernel methods
  - Boltzmann machines
  - Tensor Networks
  - Variational circuits (VQE, QAOA)
  - Quantum Neural Networks





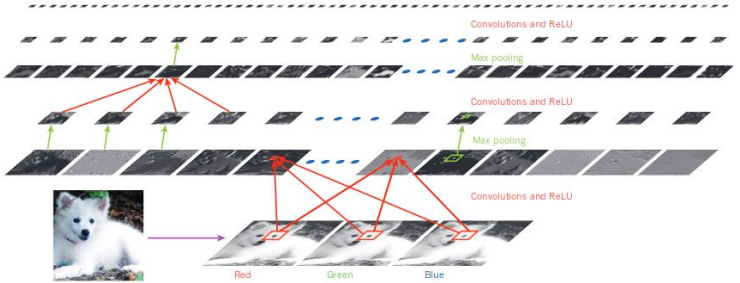



LESSONS FROM
DEEP LEARNING

Why is Deep Learning successful?

- Hardware advancements (GPUs)

- Workhorse algorithms
  (backpropagation, stochastic gradient descent)



- Specialized, user-friendly software

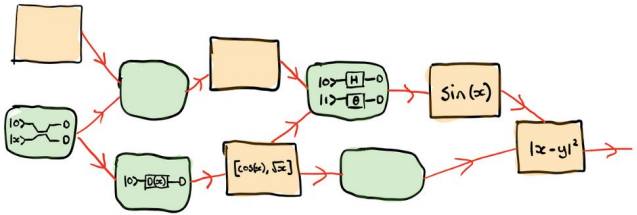
PyTorch

TensorFlow

What can we leverage?

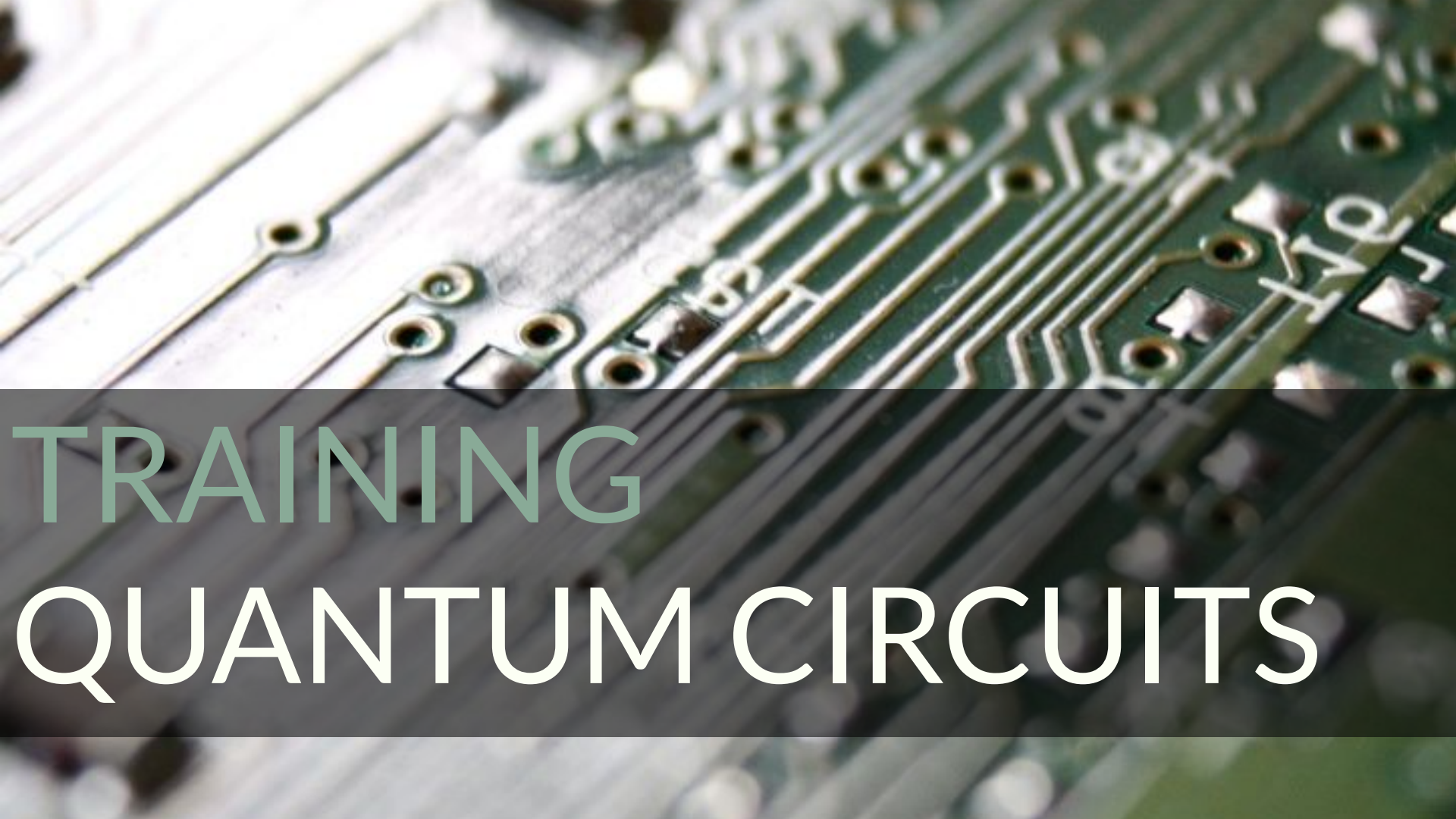
- Hardware advancements (**QPUs**)

- Workhorse algorithms
  (**quantum-aware** backpropagation, stochastic gradient descent)



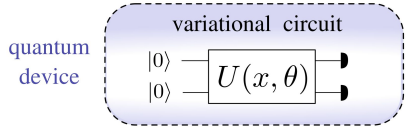
- Specialized, user-friendly software

PENNYLANE

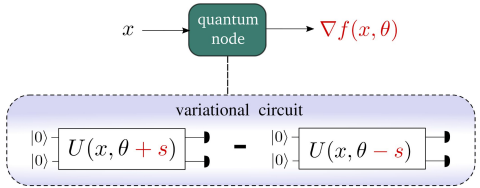

TRAINING
QUANTUM CIRCUITS

Key Concepts for QML

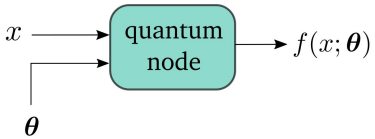
- Variational circuits



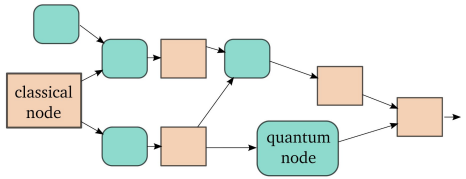
- Quantum circuit learning



- Quantum nodes

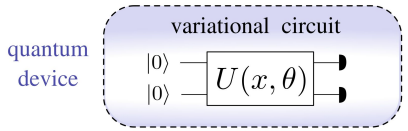


- Hybrid computation

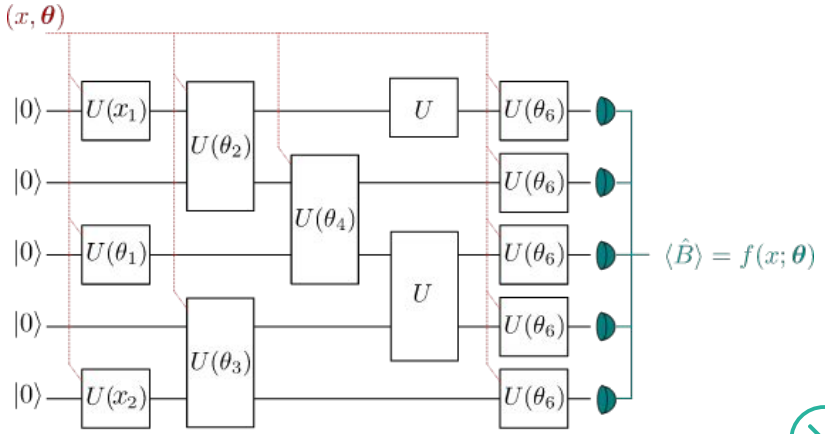


Variational Circuits

- Main QML method for near-term (NISQ) devices
- Same basic structure as other modern algorithms:
  - Variational Quantum Eigensolver (VQE)
  - Quantum Alternating Operator Ansatz (QAOA)



=



I. Preparation of a fixed initial state

II. Quantum circuit; input data and free parameters are used as gate arguments

III. Measurement of fixed observable

How to ‘train’ quantum circuits?

Two approaches:

I. *Simulator-based*

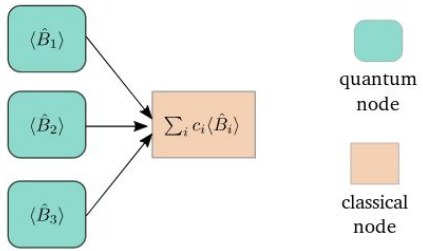
- Build simulation inside existing classical library
- Can leverage existing optimization & ML tools
- Great for small circuits, but not scalable

STRAWBERRY FIELDS

QuantumFlow

II. *Hardware-based*

- No access to quantum information; only have measurements & expectation values
- Needs to work as hardware becomes more powerful and cannot be simulated



Gradients of quantum circuits ∇f

- Training strategy: use gradient descent algorithms.
- Need to compute gradients of variational circuit outputs with respect to their free parameters.
- How can we compute gradients of quantum circuits when even simulating their output is classically intractable?

The ‘parameter shift’ trick

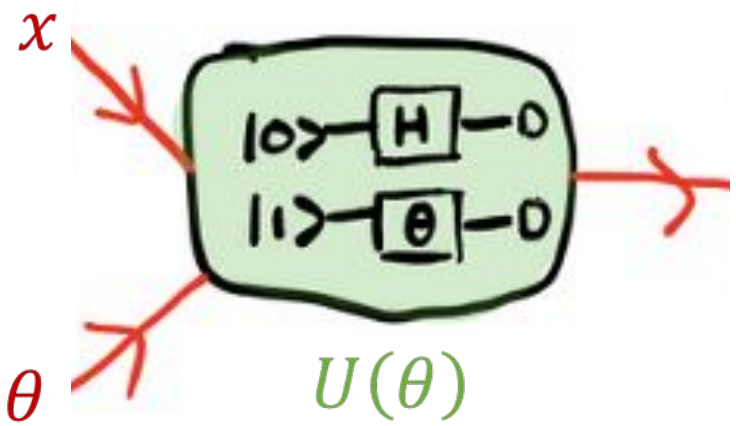
$$f(\theta) = \sin\theta \Rightarrow \partial_\theta f(\theta) = \cos\theta$$

$$\cos\theta = \frac{\sin(\theta+\pi/4)-\sin(\theta-\pi/4)}{\sqrt{2}}$$

$$\partial_\theta f = \frac{1}{\sqrt{2}}[f(\theta+\pi/4) - f(\theta-\pi/4)]$$

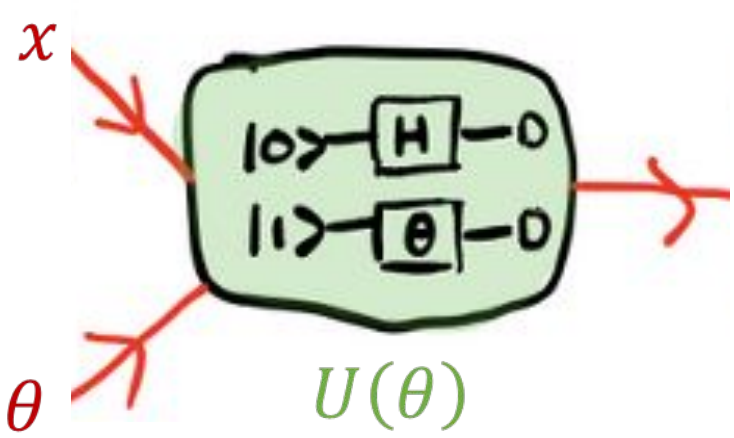
Optimization of quantum circuits

Main insight: Use the same quantum hardware to evaluate its own gradients.



$$f(x,\theta) = \langle \hat{B} \rangle$$
$$= \langle U^{\dagger}(\theta)\hat{B}\,U(\theta) \rangle$$

Parameter shift rule



$$f(\theta)$$

$$\partial_\theta f(\theta) = c[f(\theta + s) - f(\theta - s)]$$

The parameters ***c*** and ***s*** depend on the specific function. Crucially, ***s*** is large.

Quantum Circuit Learning, Physical Review A **98**, 032309 (2018)
Evaluating analytic gradients on quantum hardware, Physical Review A **99**, 032331 (2019)

This is not finite difference!

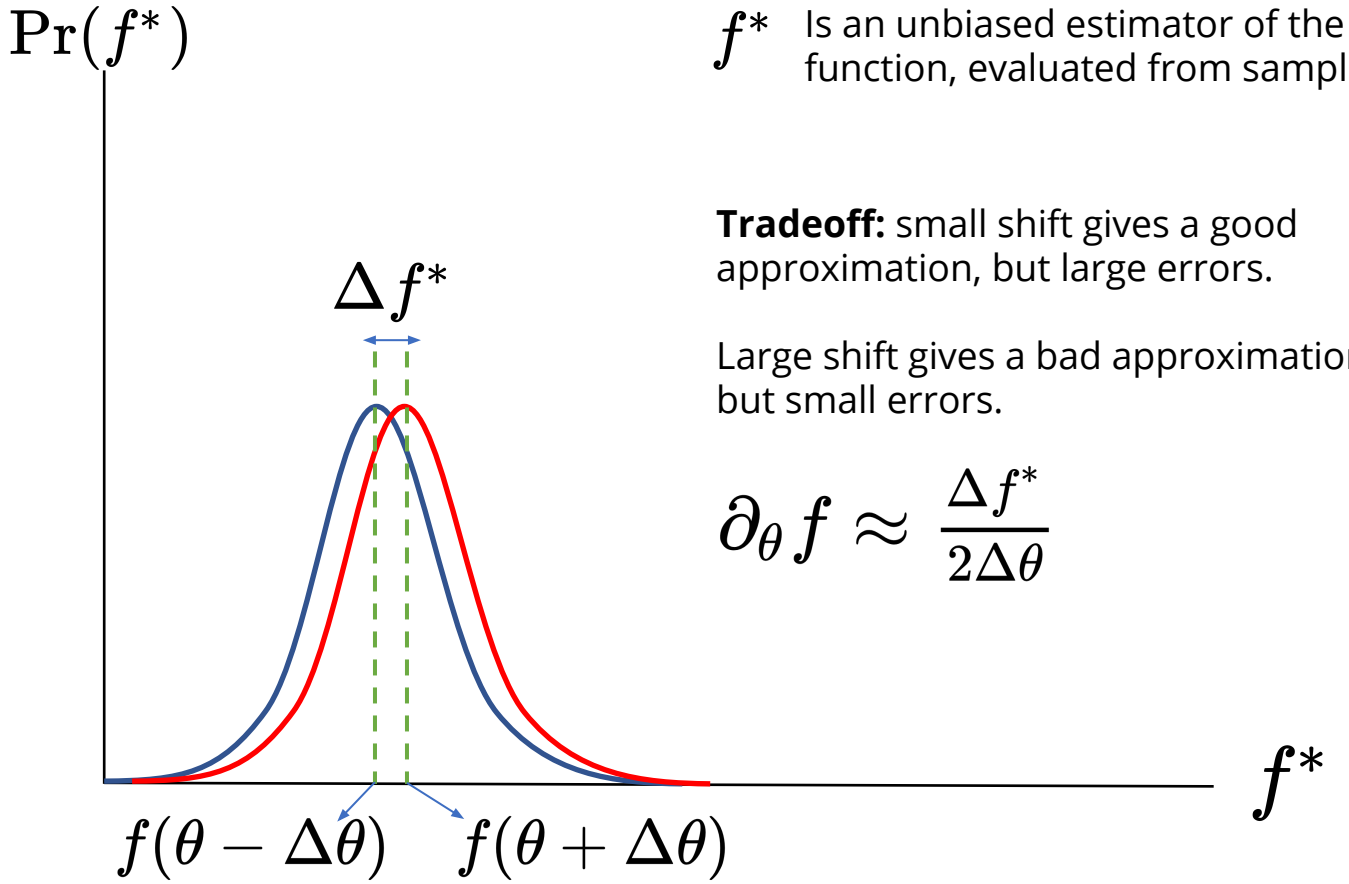
$$\partial_\theta f(\theta) = c[f(\theta + s) - f(\theta - s)]$$

- Exact
- No restriction on the shift – in general, we want a **large** shift

$$\partial_\theta f(\theta) = \frac{f(\theta+\Delta\theta) - f(\theta-\Delta\theta)}{2\Delta\theta}$$

- Only an **approximation**
- Requires that shift is small
- Known to give rise to numerical issues
- For NISQ devices, small shifts could lead to the resulting difference being swamped by noise

Finite difference



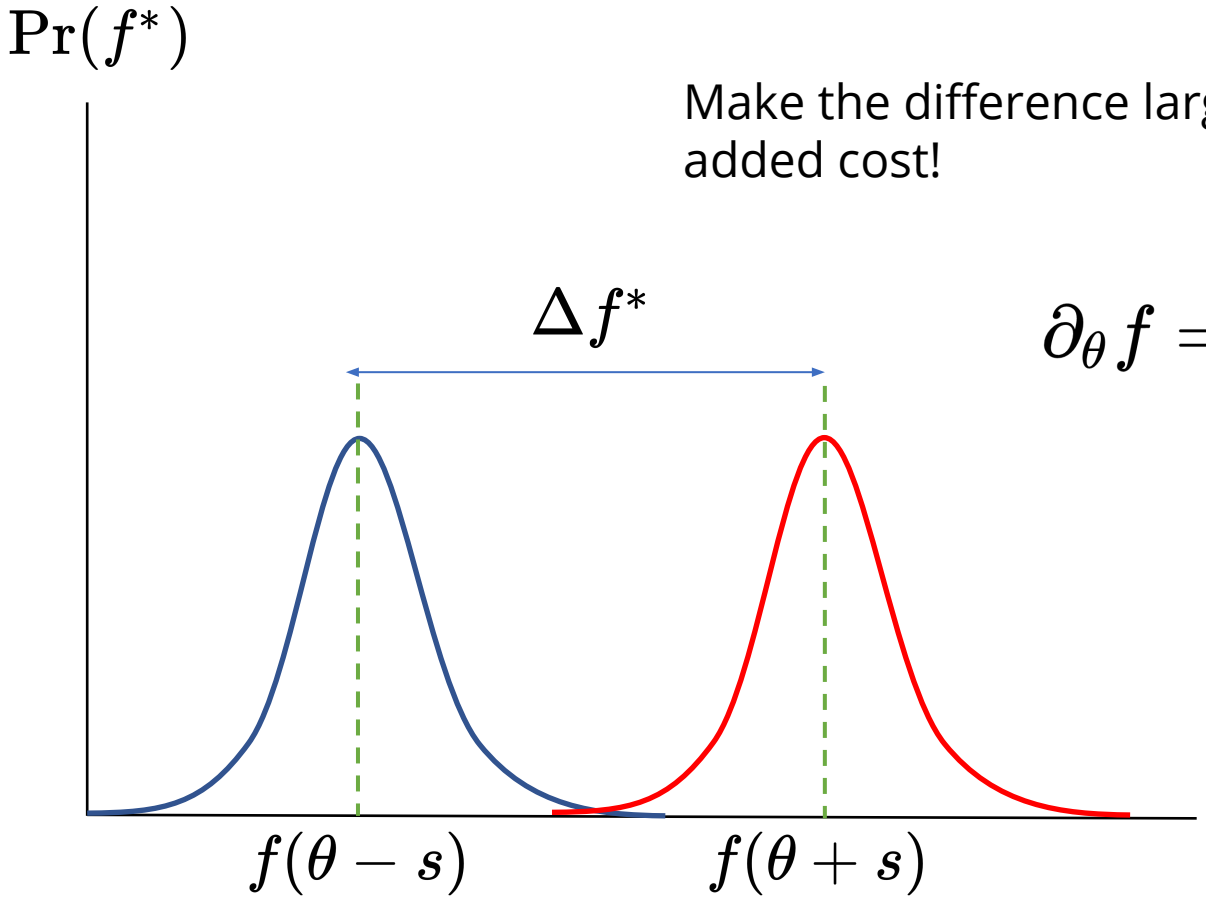
f^* Is an unbiased estimator of the function, evaluated from sampling

Tradeoff: small shift gives a good approximation, but large errors.

Large shift gives a bad approximation, but small errors.

$$\partial_\theta f \approx \frac{\Delta f^*}{2\Delta\theta}$$

Parameter shift rule

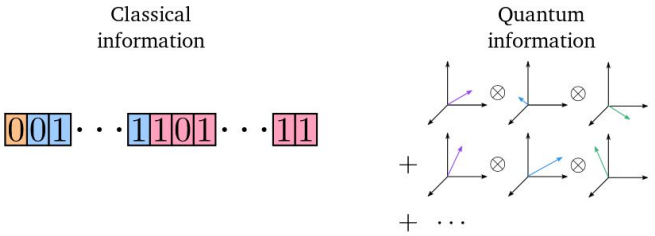


Make the difference larger at no added cost!

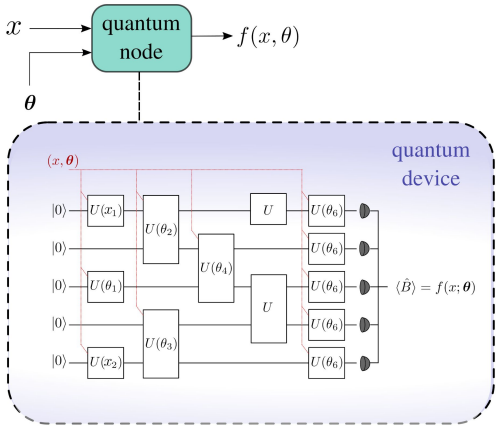
$$\partial_\theta f = c\Delta f^*$$

Quantum Nodes

- Classical and quantum information are distinct

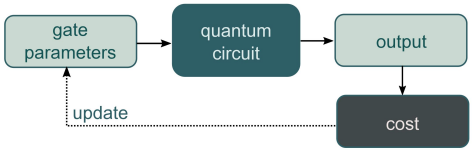


- QNode: common interface for quantum and classical devices
  - Classical device sees a callable parameterized function
  - Quantum device sees fine-grained circuit details

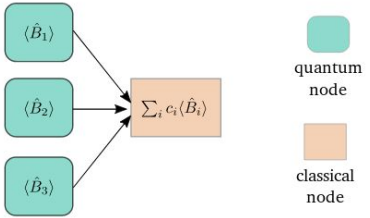


Hybrid Computation

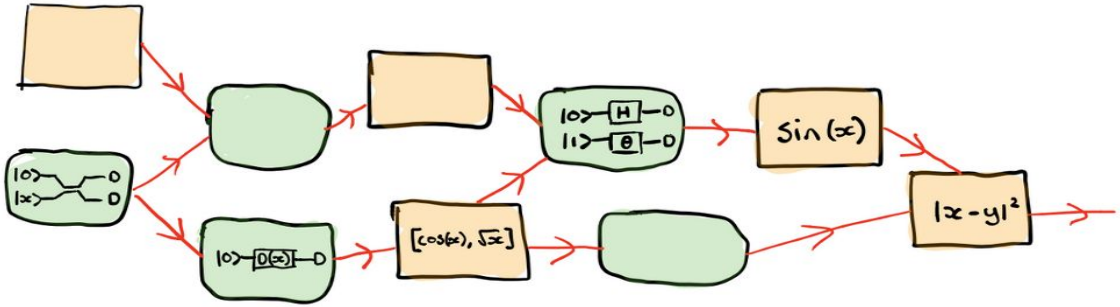
- Use QPU with classical coprocessor
  - Classical optimization loop



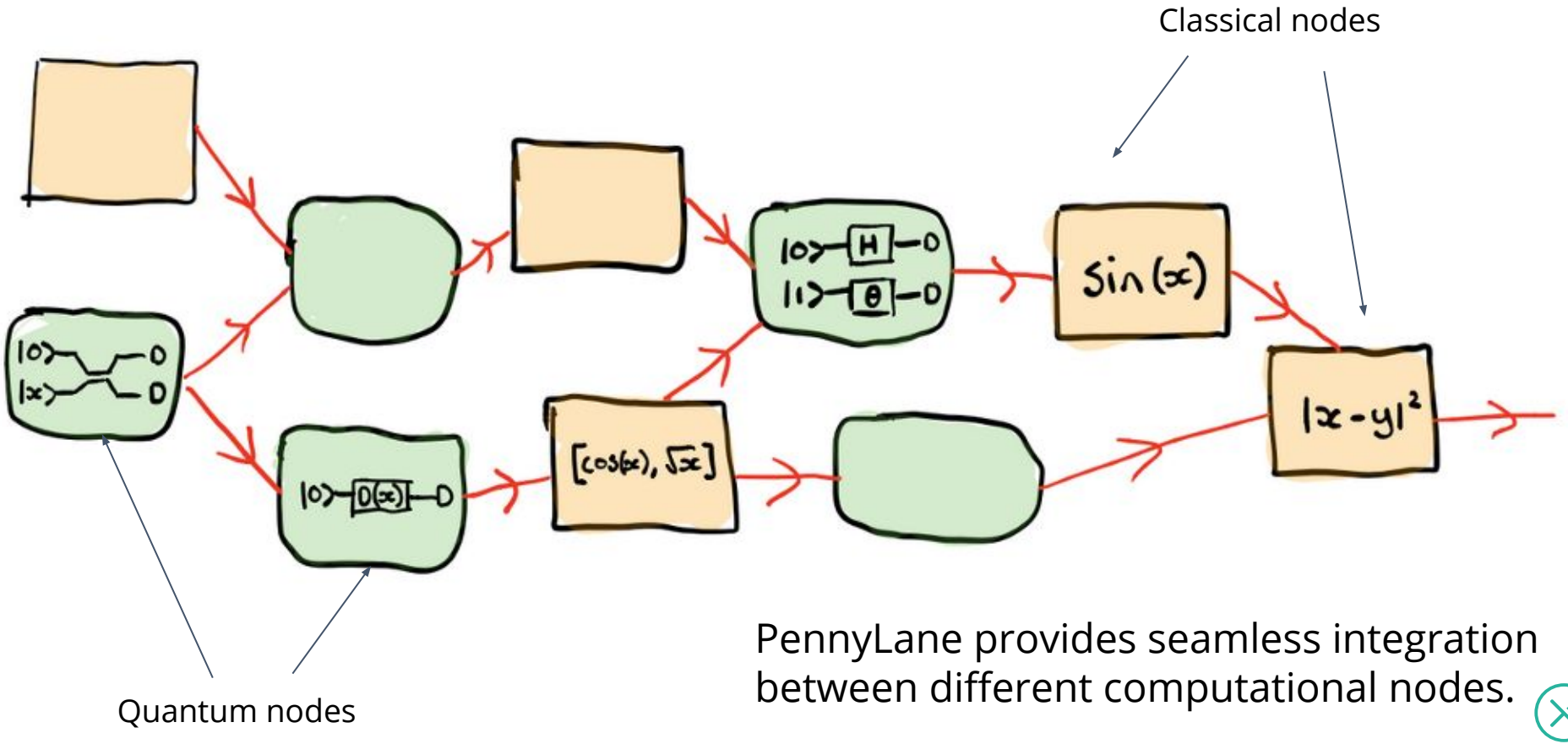
  - Pre-/post-process quantum circuit outputs



  - Arbitrarily structured hybrid computations



Hybrid computation



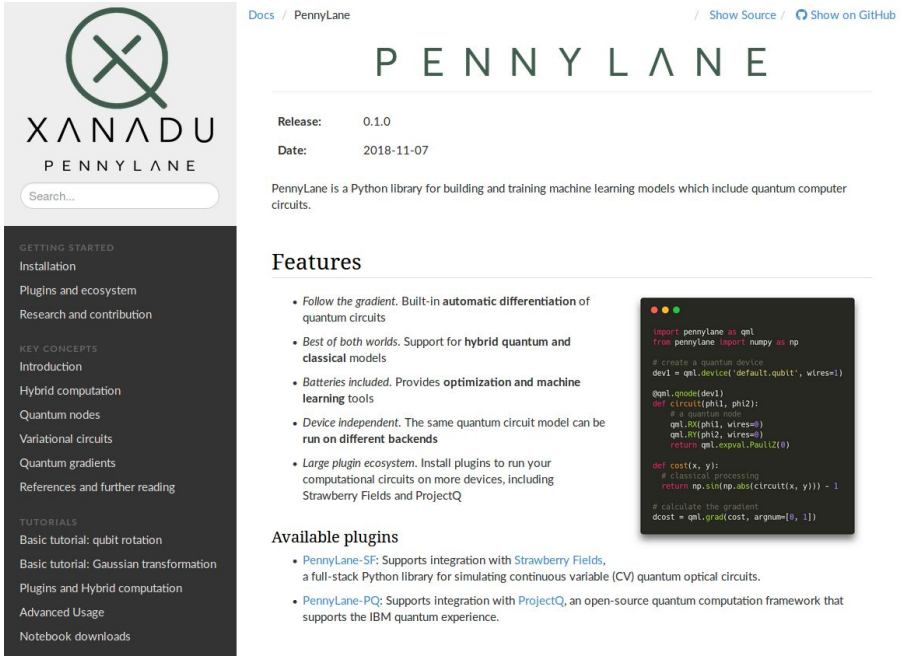
PennyLane provides seamless integration between different computational nodes.


PENNYLANE

PennyLane

"The PyTorch of quantum computing"

- Train a quantum computer the same way as a neural network
- Designed to scale as quantum computers grow in power
- Compatible with Xanadu, Rigetti, IBM, Microsoft platforms



https://github.com/XanaduAI/pennylane

https://pennylane.readthedocs.io

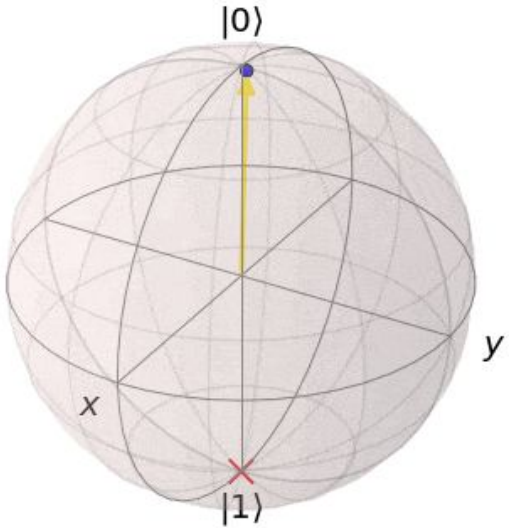
https://pennylane.ai

Comes with a growing plugin ecosystem, supporting a wide range of quantum hardware and classical software

Qubit Rotation Tutorial

The PennyLane version of “hello world”.

Goal is to build a single-qubit circuit that rotates a qubit to a desired pure state.



Import libraries

```
import pennylane as qml
from pennylane import numpy as np
```

Important: NumPy must be imported from PennyLane to ensure compatibility with automatic differentiation.

Basically, this allows you to use NumPy as usual.

You can also use **PyTorch** or **TensorFlow** instead of NumPy.

Create device

Device name

Wires are subsystems (because they are represented as wires in a circuit diagram)

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

Any computational object that can apply quantum operations and return a measurement result is called a quantum **device**.

In PennyLane, a device could be a hardware device (such as the Rigetti QPU, via the PennyLane-Forest plugin), or a software simulator (such as Strawberry Fields, via the PennyLane-SF plugin).

Create a qnode

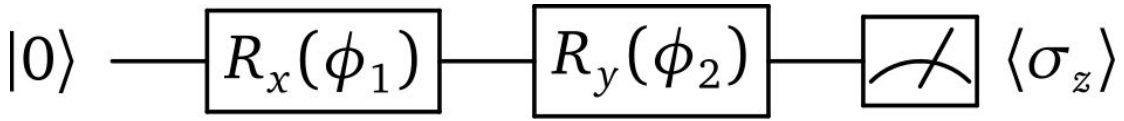
Rotation parameters

```
@qml.qnode(dev)
def circuit(params):
    qml.RX(params[0], wires=0)
    qml.RY(params[1], wires=0)
    return qml.expval(qml.PauliZ(0))
```

Python decorator

Wire the gate acts on

Expectation value output

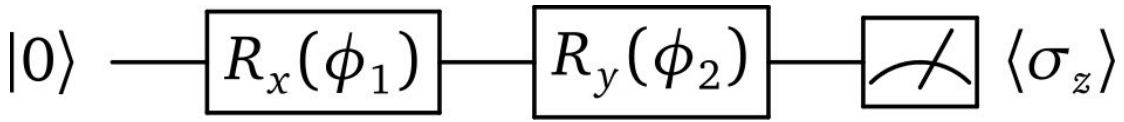


QNodes are quantum functions, described by a quantum circuit. They are bound to a particular quantum device, which is used to evaluate expectation values of this circuit.

Decorator

Python decorator

```
@qml.qnode(dev)
def circuit(params):
    qml.RX(params[0], wires=0)
    qml.RY(params[1], wires=0)
    return qml.expval(qml.PauliZ(0))
```



`@qml.qnode(dev)` is a python *decorator*, whose role is to wrap a function and change its properties. In this case, the decorator is used to convert the defined circuit into a qnode object.

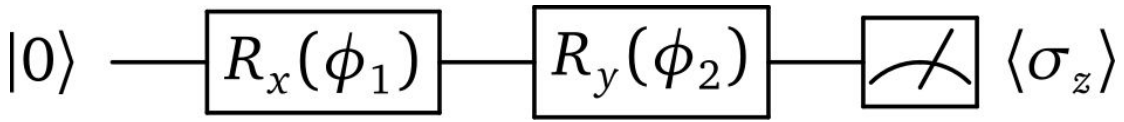
Specify circuit

Rotation parameters

```
@qml.qnode(dev)
def circuit(params):
    qml.RX(params[0], wires=0)
    qml.RY(params[1], wires=0)
    return qml.expval(qml.PauliZ(0))
```

Wire the gate acts on

Expectation value output



The circuit is applying two rotations on the qubit, then returning the expectation value of the Pauli Z operator.

Cost function

```
def cost(params):
    expval = circuit(params)
    return np.abs(expval - (-1)) ** 2
```

Can define any differentiable NumPy function from the output of the qnode.

In this case, we want the expectation value of the circuit to be -1.

Initial parameters

```
params = np.random.normal(size=(2,))
circuit(params)
```

Dimension of `params`

We can evaluate the circuit at any value of `params`

In this case, there are two rotation angles, which we initialize randomly from the standard normal distribution.

When any qnode is evaluated, PennyLane calls the device itself to obtain the result.

Optimize the circuit

```
opt = qml.AdamOptimizer()
steps = 300

for i in range(steps):
    params = opt.step(cost, params)

print('Circuit output:', circuit(params))
print('Final parameters:', params)
```

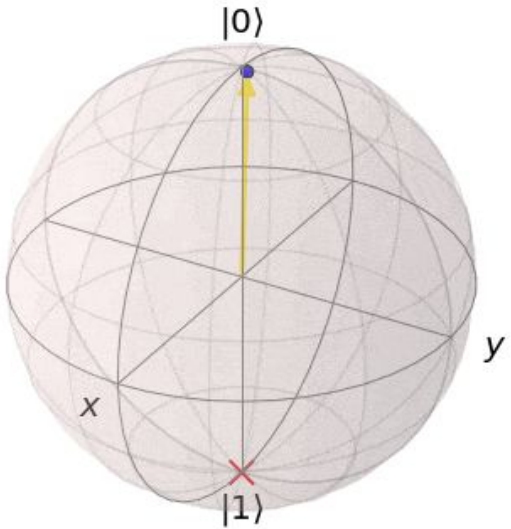
Improves parameters by gradient descent

We can choose from a wide variety of gradient-based optimizers. In this case we select the Adam optimizer.

The parameters are optimized one step at a time for a total of 300 steps, then printed.

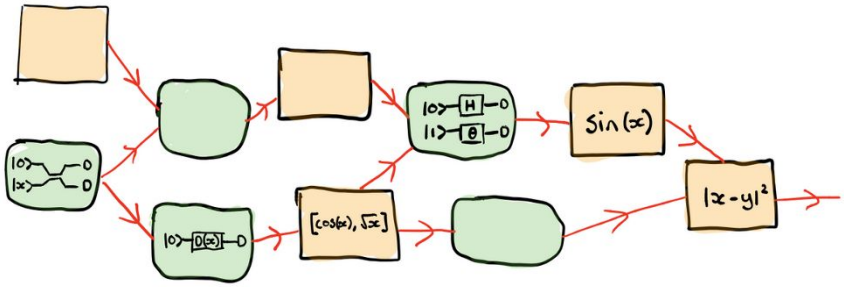
Putting it all together

```
import pennylane as qml
from pennylane import numpy as np

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

@qml.qnode(dev)
def circuit(params):
  qml.RX(params[0], wires=0)
  qml.RY(params[1], wires=0)
  return qml.expval(qml.PauliZ(0))

def cost(params):
  expval = circuit(params)
  return np.abs(expval - (-1)) ** 2

params = np.random.normal(size=(2,))

opt = qml.AdamOptimizer()
steps = 300

for i in range(steps):
    params = opt.step(cost, params)

print('Circuit output:', circuit(params))
print('Final parameters:', params)
```



Summary

- Run and optimize directly on quantum Hardware (GPU→QPU)
- “Quantum-aware” implementation of gradient descent optimization
- Hardware agnostic and extensible via plugins
- Open-source and extensively documented
- Use-cases:
    - Machine learning on large-scale quantum computations
    - Hybrid quantum-classical machine learning



https://github.com/XanaduAI/pennylane

https://pennylane.readthedocs.io

https://pennylane.ai


Thank You
XANADU