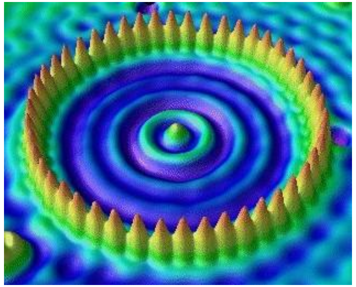
Quantum Machine Learning

Nathan Killoran

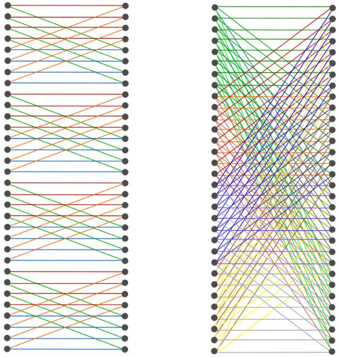
XANADU

Quantum computers are good at:

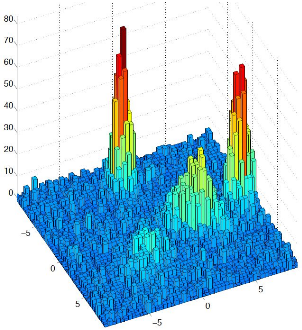
Quantum physics



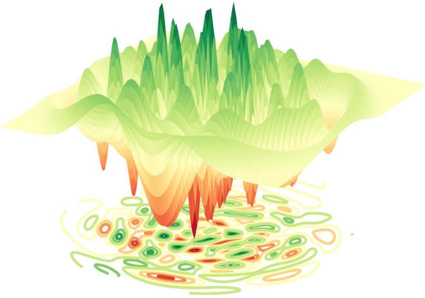
Linear algebra



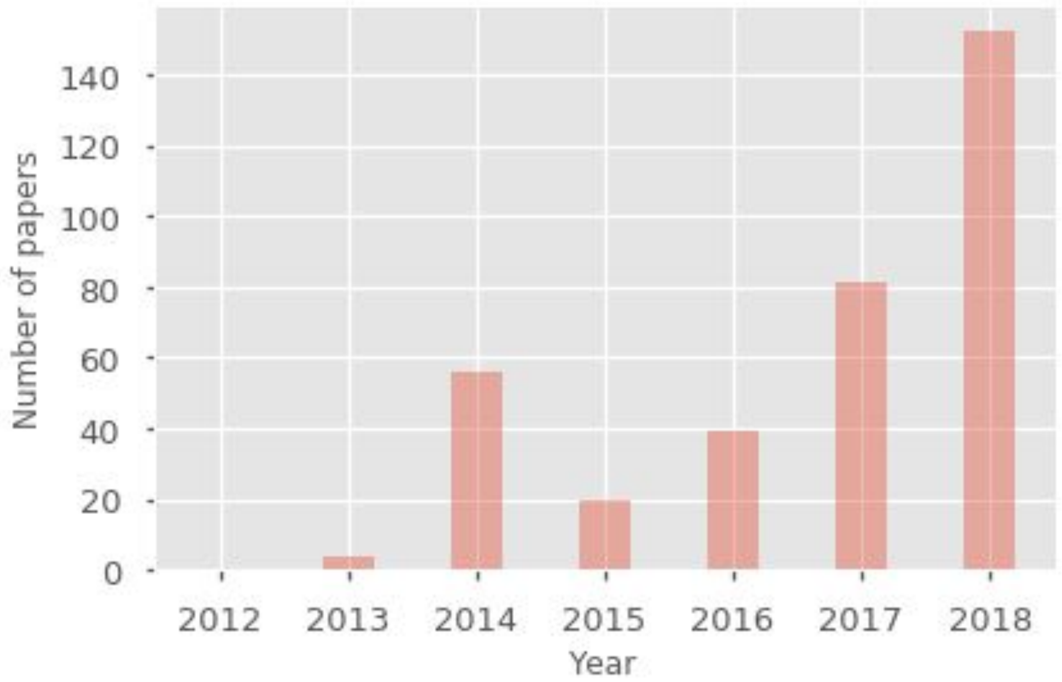
Sampling



Optimization

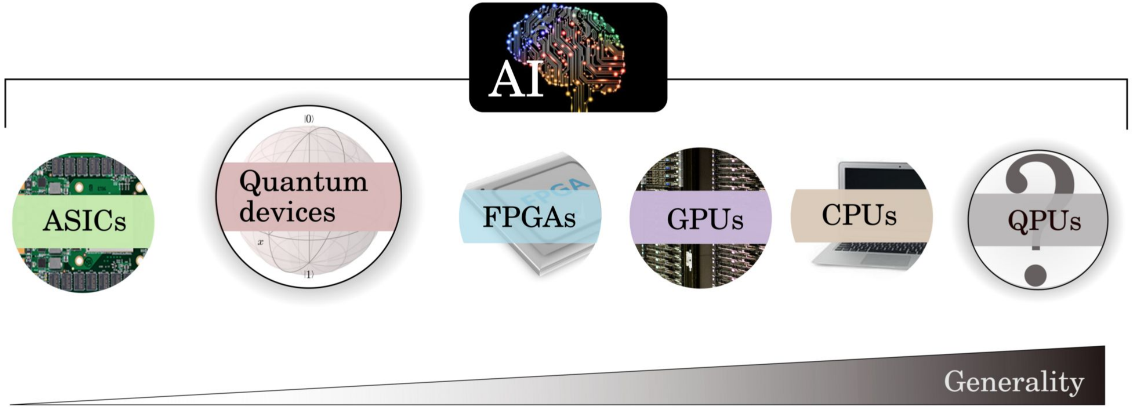


Quantum Machine Learning papers



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

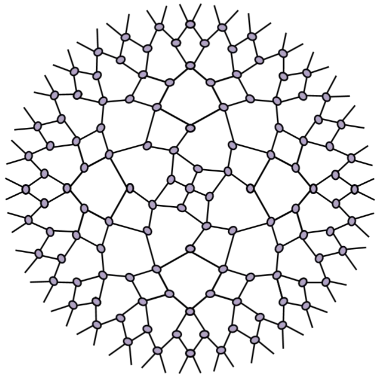


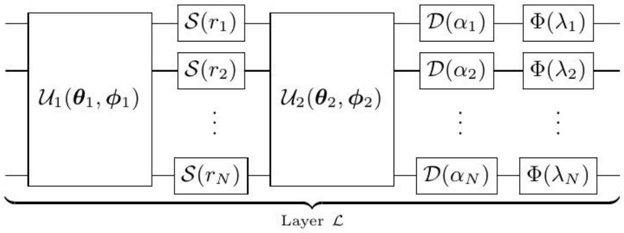
New AI models

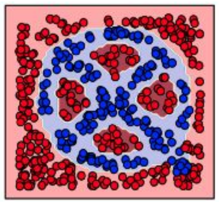
- Quantum computing can also lead to new machine learning models

- Examples currently being studied are:

  - Kernel methods
  - Boltzmann machines
  - Tensor Networks
  - Variational circuits
  - Quantum Neural Networks





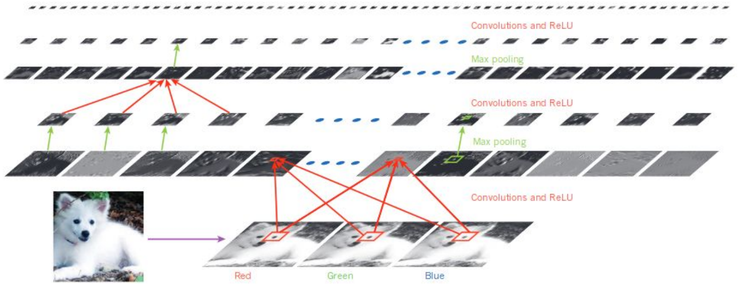



LESSONS FROM
DEEP LEARNING

Why is Deep Learning successful?

- Hardware advancements (GPUs)

- Workhorse algorithms
  (backpropagation, stochastic gradient descent)

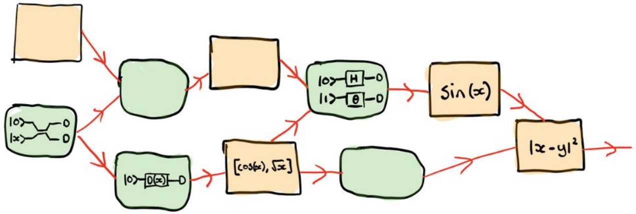


- Specialized, user-friendly software

PyTorch TensorFlow

What can we leverage?

- Hardware advancements (GPUs **+ QPUs**)



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

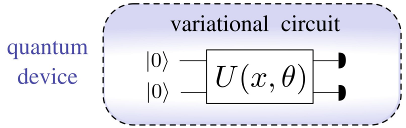
- Specialized, user-friendly software

PENNYLANE

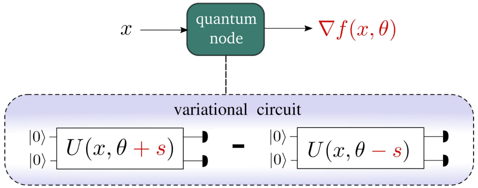
TRAINING QUANTUM CIRCUITS

Key Concepts for QML

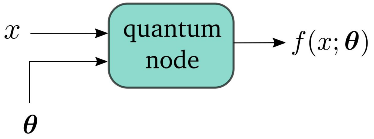
- Variational circuits



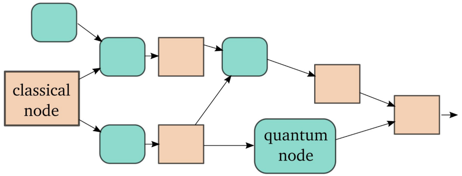
- Quantum circuit learning



- Quantum nodes



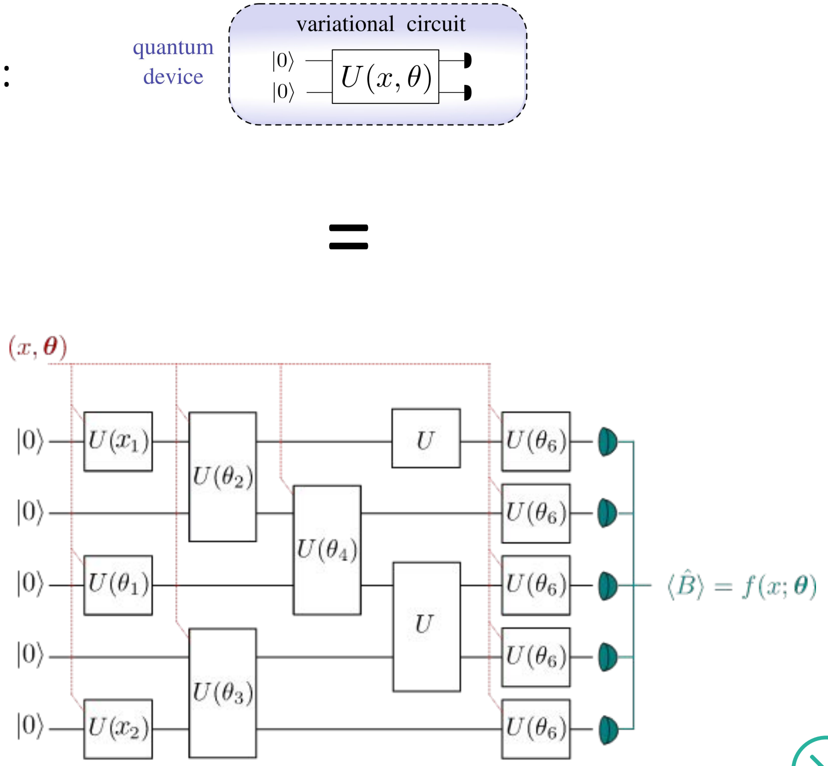
- Hybrid computation



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?

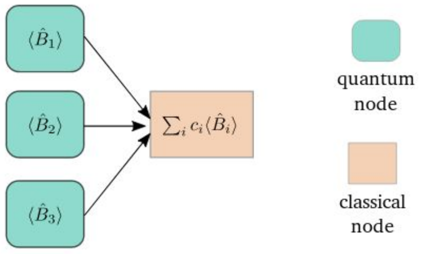
Two approaches:

I. *Simulator-based*

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

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 w.r.t. their free parameters.
- How can we compute gradients of quantum circuits when even simulating their output is classically intractable?

The ‘parameter shift’ trick

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

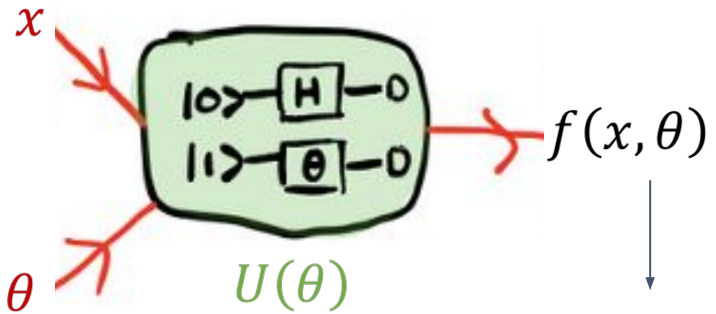
$$\cos\theta = \frac{\sin\left(\theta + \frac{\pi}{4}\right) - \sin\left(\theta - \frac{\pi}{4}\right)}{\sqrt{2}}$$

$$\partial_\theta f = \frac{1}{\sqrt{2}}\left(f\left(\theta + \frac{\pi}{4}\right) - f\left(\theta - \frac{\pi}{4}\right)\right)$$

Quantum Circuit Learning

- Use the same device to compute a function and its gradient

  - “Parameter shift” differentiation rule: gives **exact gradients**



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

  - Minimal overhead to compute gradients vs. original circuit
  - Optimize circuits using **gradient descent**
  - Compatible with classical backpropagation: hybrid models are **end-to-end differentiable**

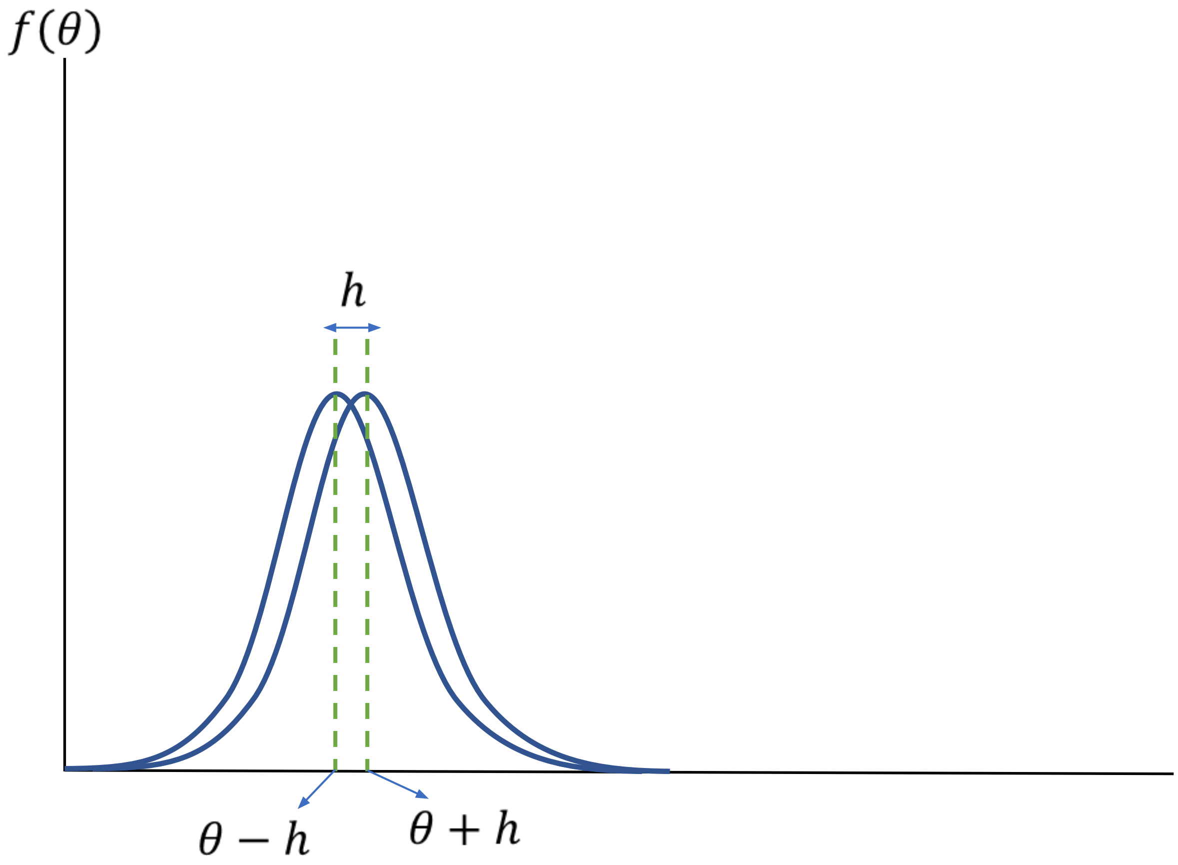
Note:
This is not finite differences!

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

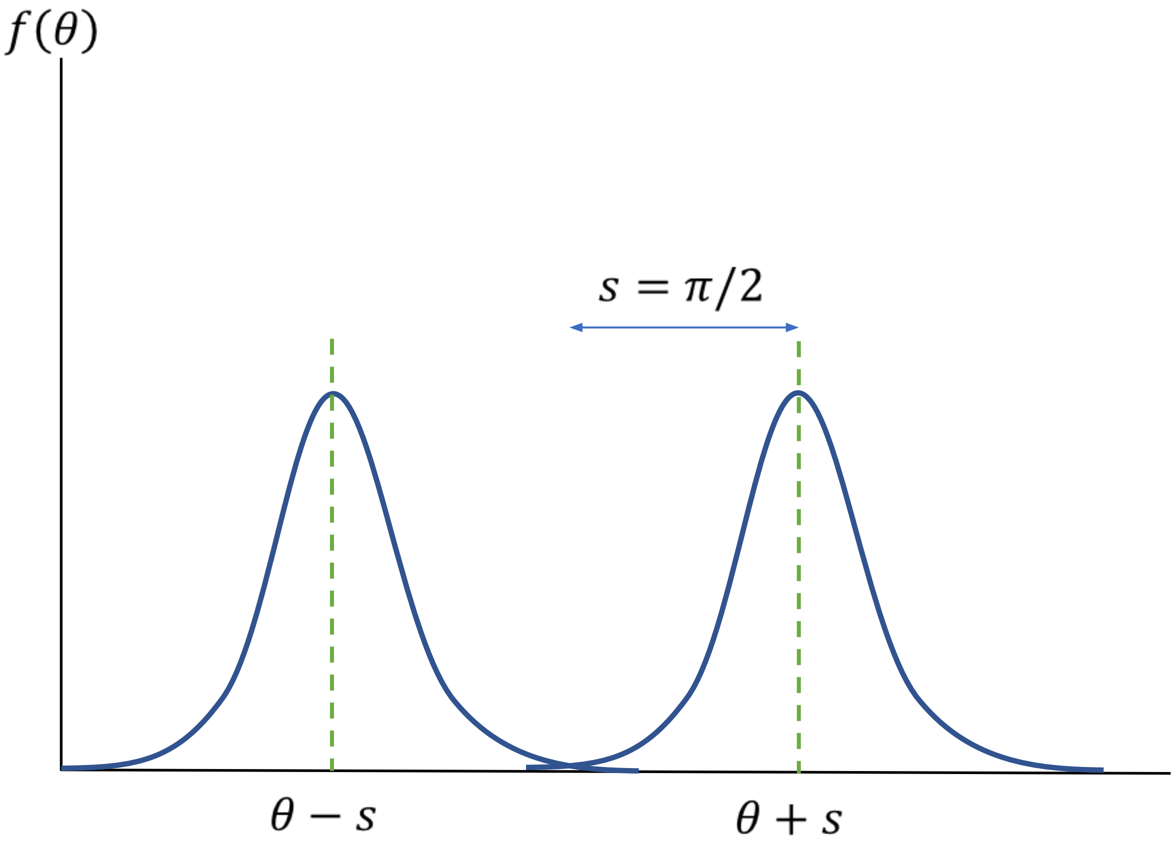
- *Exact*
- No restriction on the shift – in general, we want a *macroscopic* shift

$$\partial_\theta f(\theta) \approx \frac{f(\theta + h) - f(\theta - h)}{2h}$$

- Only an *approximation*
- Requires that h is small
- In subject to the quirks of numerical differentiation – stability, rounding error, truncation error
- For NISQ devices, small h could lead to the difference being swamped by noise

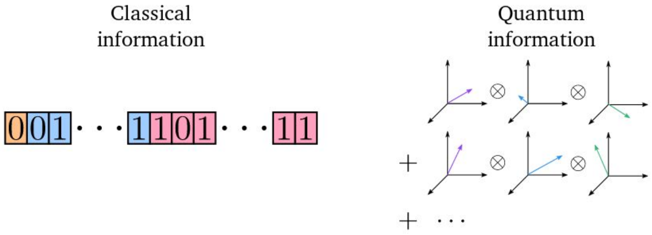


$f(\theta)$
h
$\theta - h$
$\theta + h$

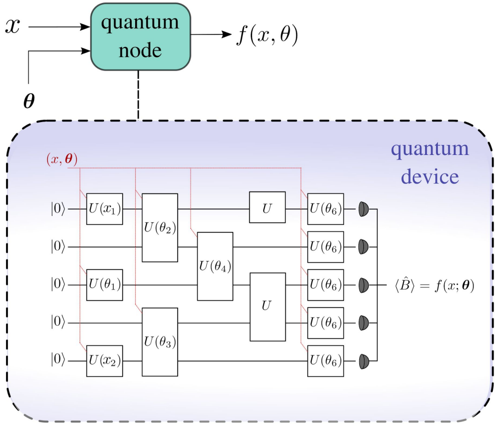

$f(\theta)$
$s = \pi/2$
$\theta - s$
$\theta + s$

Quantum Nodes

- Classical and quantum information are distinct

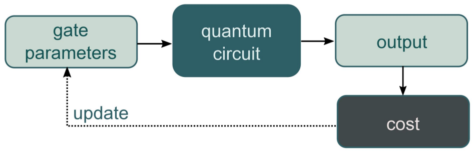


- QNode: common interface for quantum and classical devices
    - Classical device sees a callable parameterized function
    - Quantum device sees fine-grained circuit details

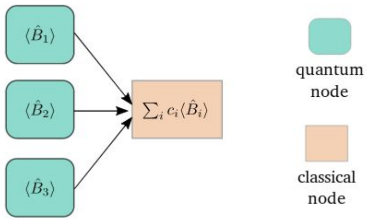


Hybrid Computation

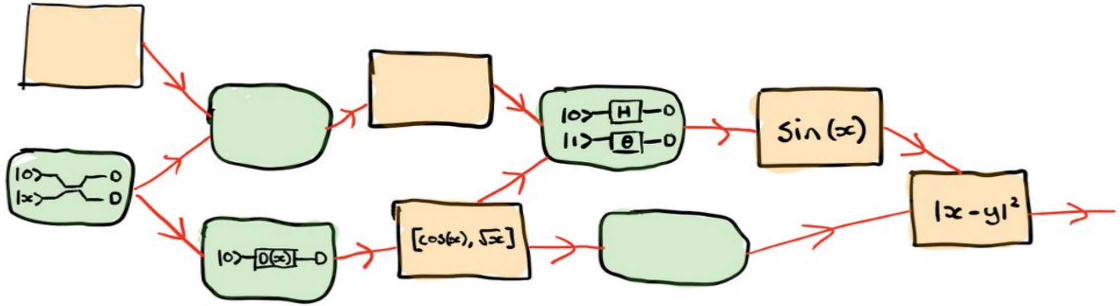
- Use QPU with classical coprocessor
  - Classical optimization loop



  - Pre-/post-process quantum circuit outputs



  - Arbitrarily structured hybrid computations

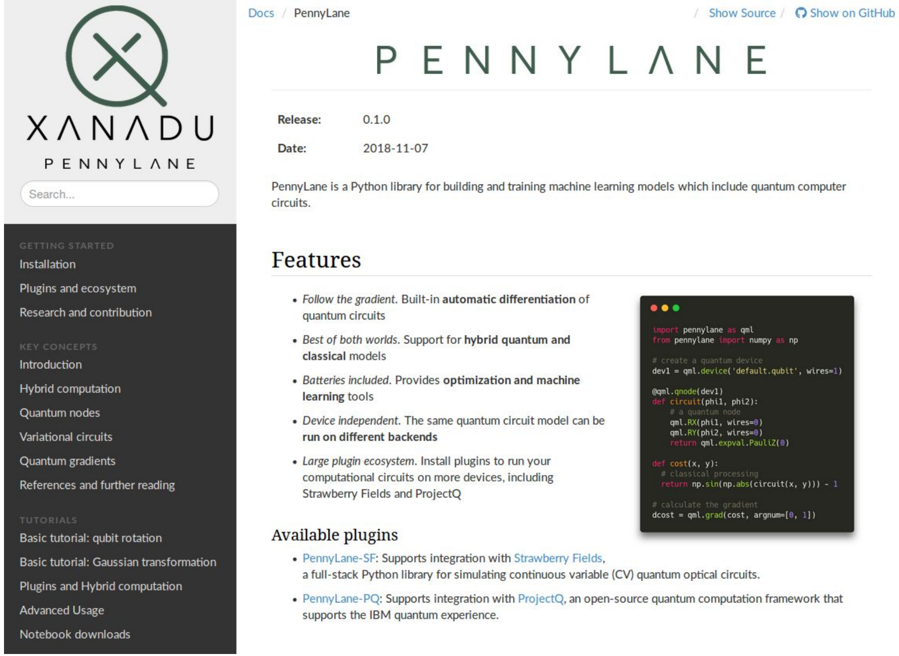



PENNYLANE

PennyLane

"The TensorFlow 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, IBM, Rigetti, and Microsoft platforms



https://github.com/XanaduAI/pennylane

https://pennylane.ai

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

PENNYLANE

PyTorch

TensorFlow

STRAWBERRY FIELDS

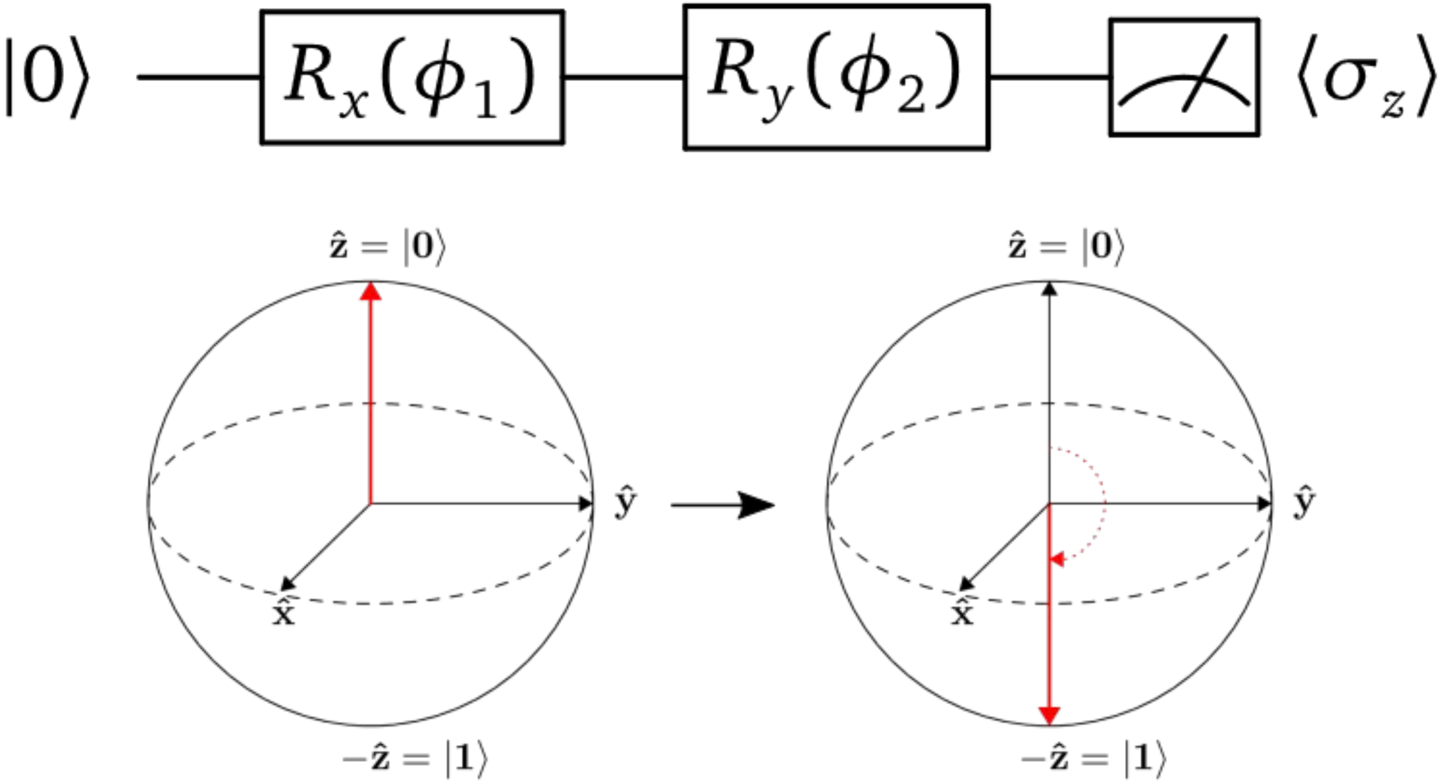
rigetti Forest

Qiskit

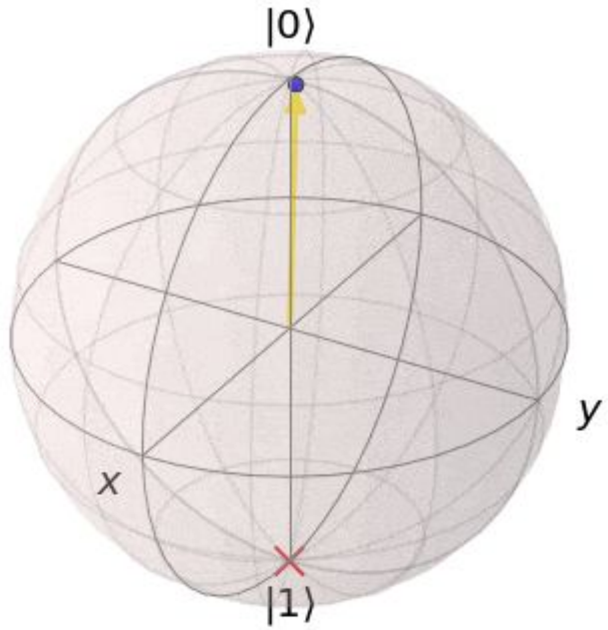
NumPy

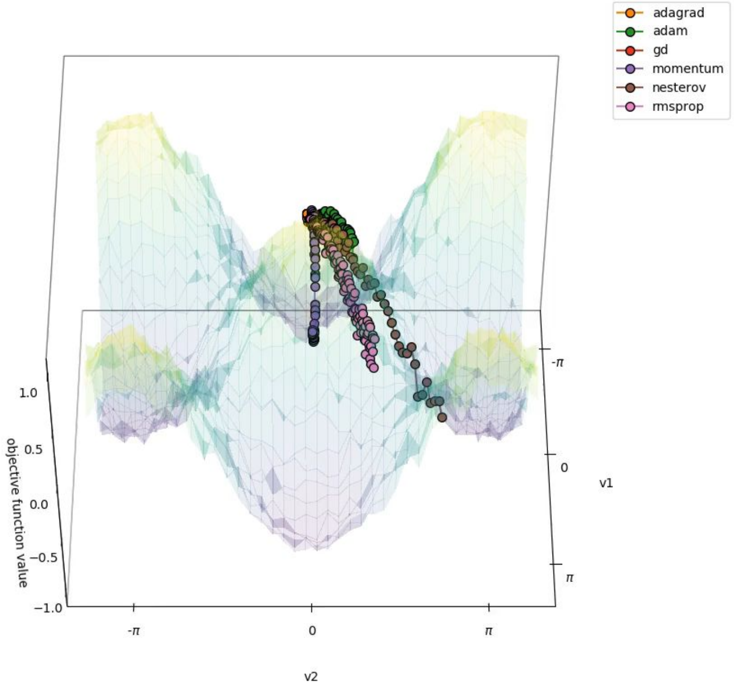
Microsoft Q#

PennyLane Example



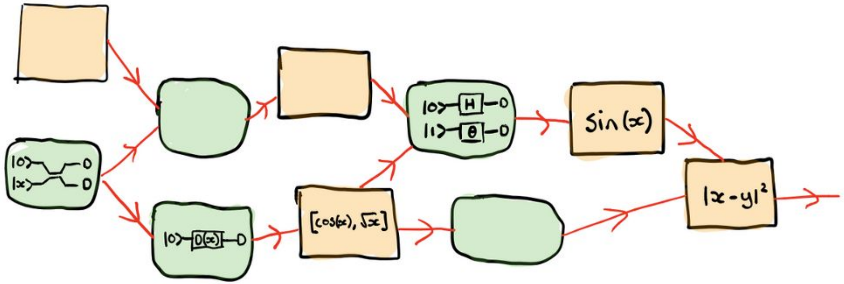
PennyLane Example





PennyLane Summary

- Run and optimize directly on quantum hardware (GPU→QPU)
- “Quantum-aware” implementation of backpropagation
- 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.ai

XANADU

Quantum Software Competition

EDUCATION AWARD | SOFTWARE AWARD | RESEARCH AWARD

A competition — **with prizes of up to $1000 on offer** — encouraging the use of quantum software across three areas: education, software development, and research.