NUS National University of Singapore

Centre for Quantum Technologies

Angelakis group

(incomplete) Introduction to Quantum Machine Learning

Supanut, CQT (26/04/2021)

About this talk:

Part 1: QML can mean different things (Big picture)

Part 2: Quantum hardware as ML models

Part 3: Quantum Kernel and Importance of data embeddings

Part 1: QML can mean different things (Big picture)

QML = Quantum + Machine Learning

1. Quantum-enhanced ML algorithms
2. Quantum computers as ML models

} Quantum computers to do some ML

3. ML for solving quantum problems
4. Quantum-inspired ML algorithms

} Classical ML for some quantum

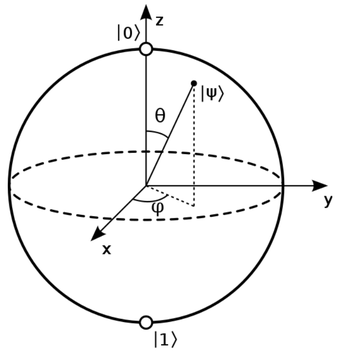
Etc.

Quantum Computers

What is QC ?

 or

Classical bit (0 or 1)



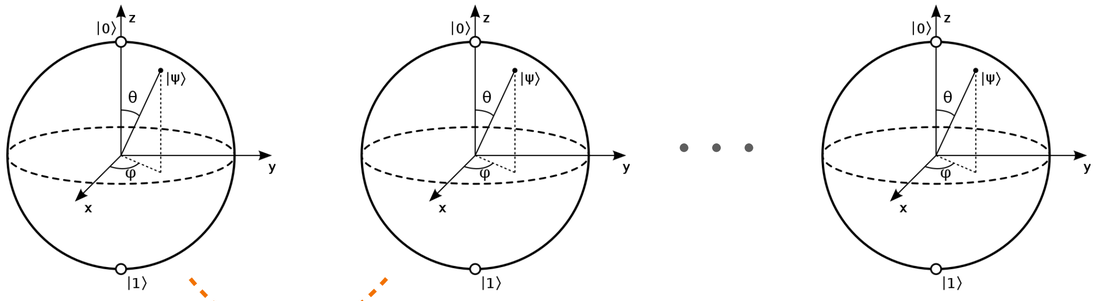
Qubit

(two-level quantum system)

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

e.g. $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

Multiple qubits



n qubits

↓

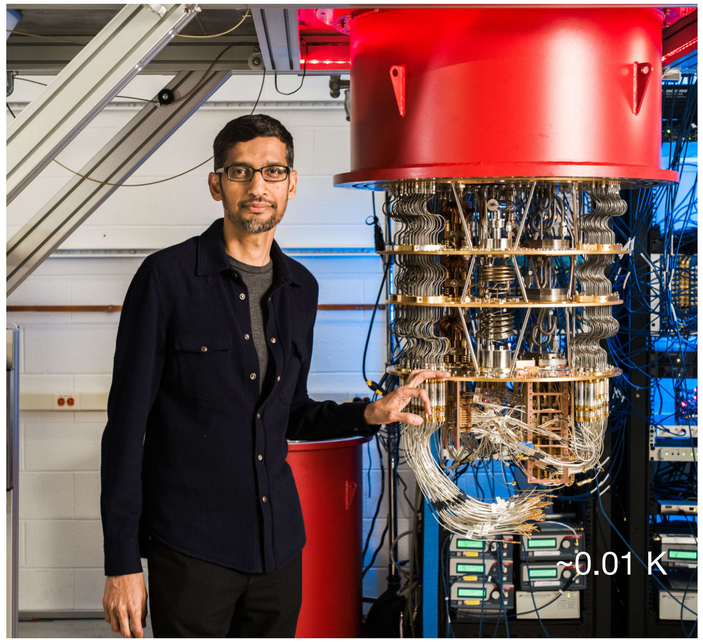
Complex vector space of 2^n **dimensions**

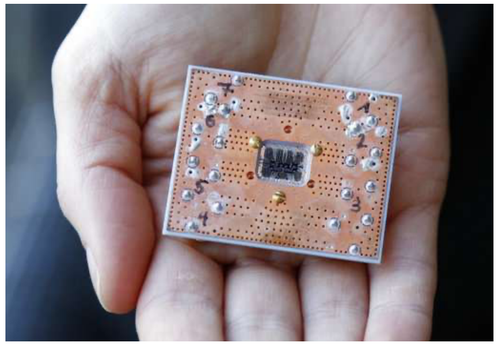
(scaling **exponential** in number of qubits !)

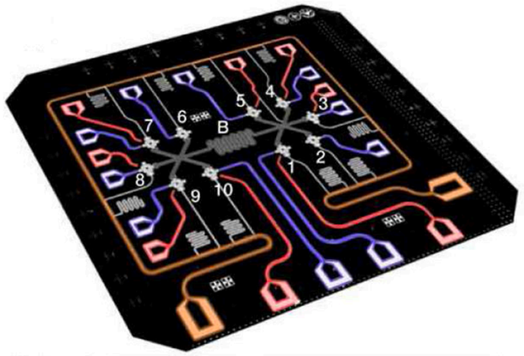
Quantum Computers

Implementation

Superconducting qubits:





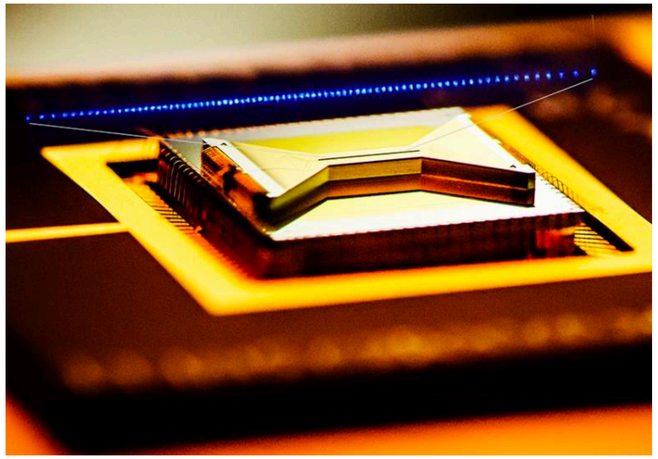


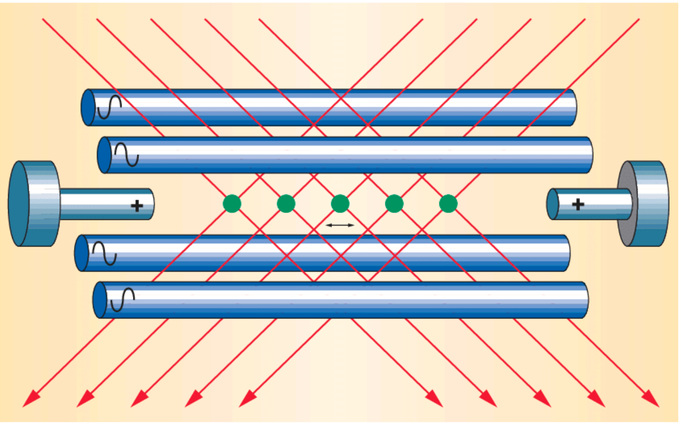
Visit for cool animations :

https://www.rigetti.com/

https://ionq.com/technology

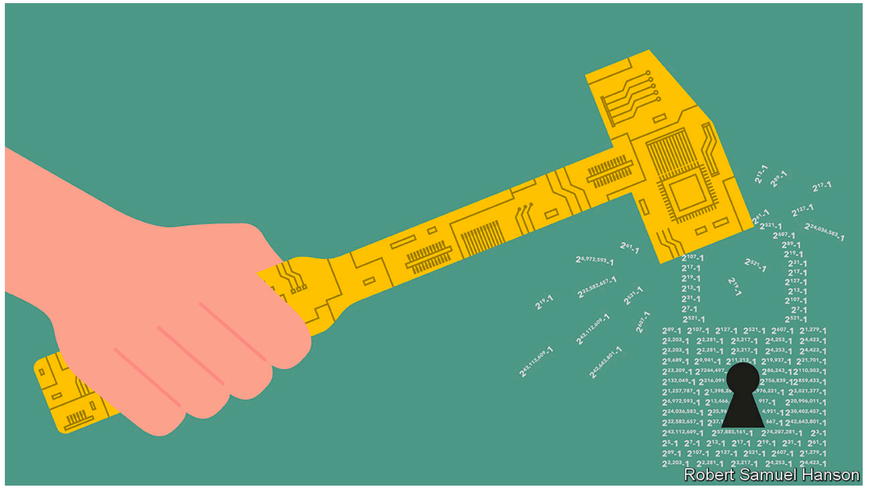
Trapped-ion qubits:





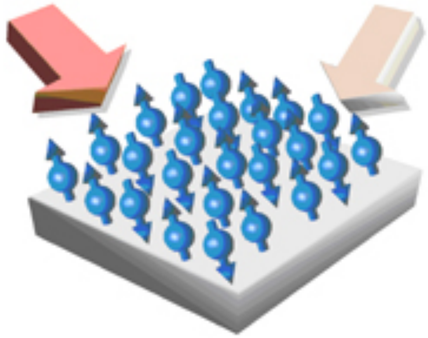
Why Quantum Computers ?

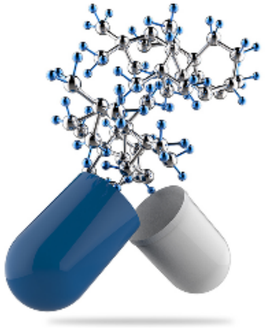
Breaking security (RSA with Shor's)



Simulation of other quantum systems

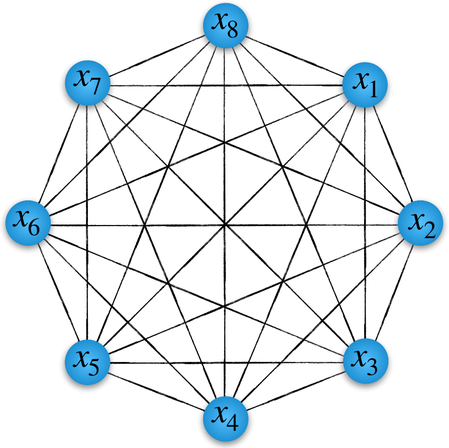
e.g. molecules, material science, drug design

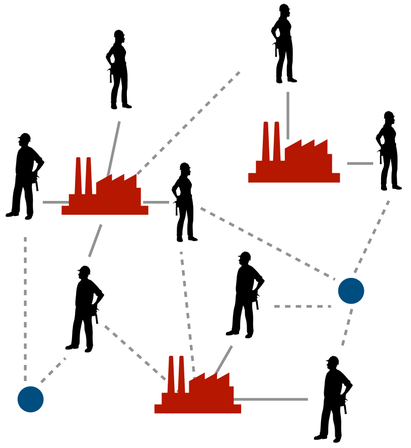




Combinatorial Optimisation

e.g. facility allocation, travelling salesman





Quantum Enhanced ML algorithms

Translate existing algorithms into quantum version ! e.g. $A\vec{x} = \vec{b} \rightarrow \hat{A}\,|x\rangle = |b\rangle$

Intuitions

$\vec{x} \in \mathcal{C}^N$

(input with N features)

Classically, use $O(N)$ **bits**

qRAM

(amplitude encoding)

$$|x\rangle = \sum_i^N x_i\,|i\rangle \in \mathcal{C}^N$$

(quantum state in N dimensions)

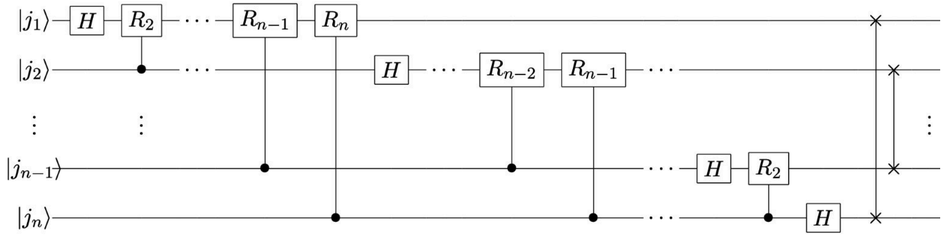
Quantumly, use $n = \log_2 N$ **qubits**

Natural quantum dynamics: $|\psi\rangle = \hat{U}\,|x\rangle$ [matrix multiplication of **size** N with $\log_2 N$ **qubits**]

evolve **efficiently** and **precisely**

= quantum speed up (much less number of operations !)

Examples: Basic Linear Algebra Subroutines (BLAS)



Classical

Fast Fourier transforms, $O(N \log_2 N)$

Eigenvectors and values, $O(N^2)$

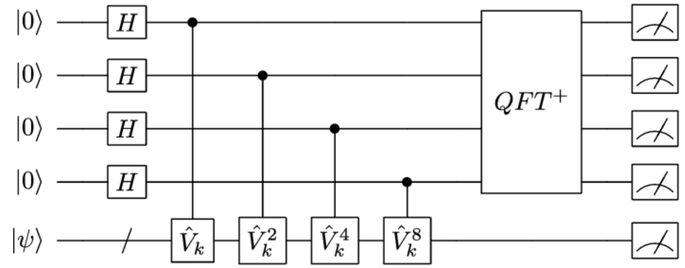
Solving linear equations, $O(N \log_2 N)$

Quantum

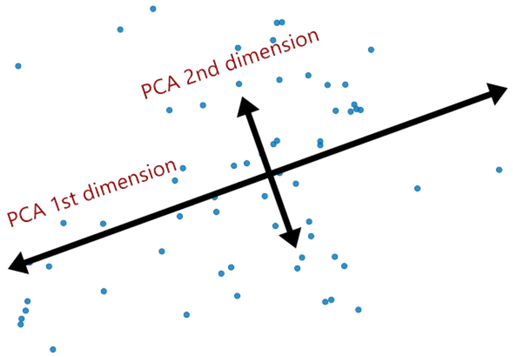
QFT, $O((\log_2 N)^2)$

Q. phase estimation, $O((\log_2 N)^2)$

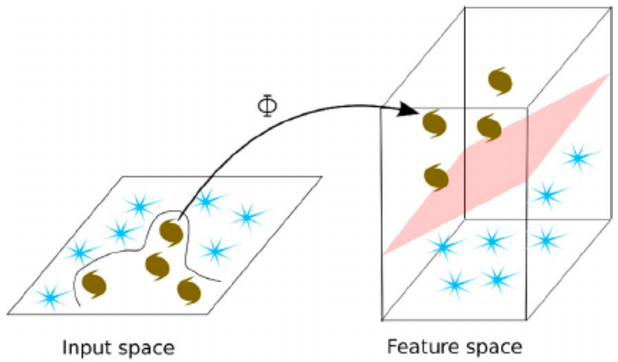
HHL algorithms, $O((\log_2 N)^2)$

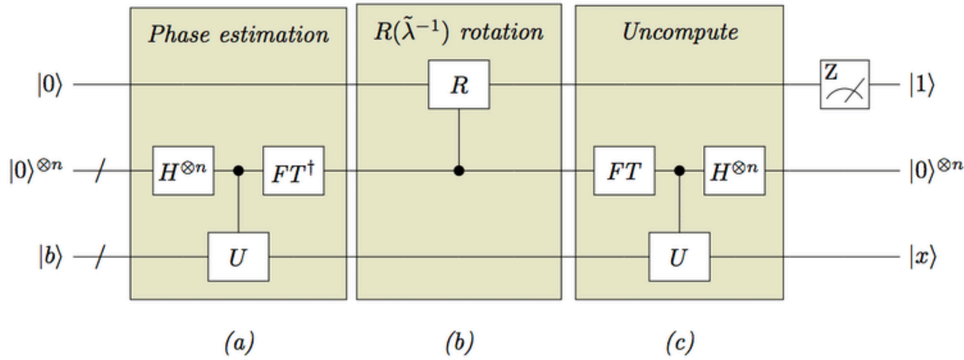


Quantum PCA



Quantum SVM





Focus on **less operations**

Quantum Enhanced ML algorithms

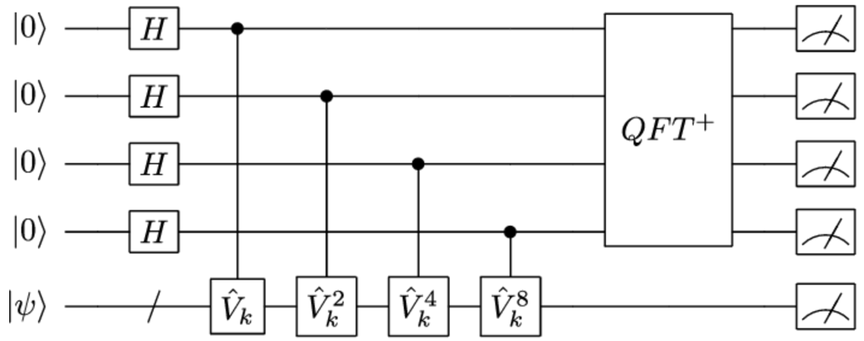
Challenges

Require **qRAM !**

$$\overrightarrow{x} \in \mathcal{C}^N \xrightarrow{\text{qRAM}} |x\rangle = \sum_i^N x_i |i\rangle \in \mathcal{C}^N$$

$$|\psi\rangle = \hat{U}|x\rangle$$

evolve **efficiently** and **precisely**



Require **error corrections !!**

Input and output problems

ML for quantum problems

CONTENTS

Review: https://arxiv.org/abs/1903.10563

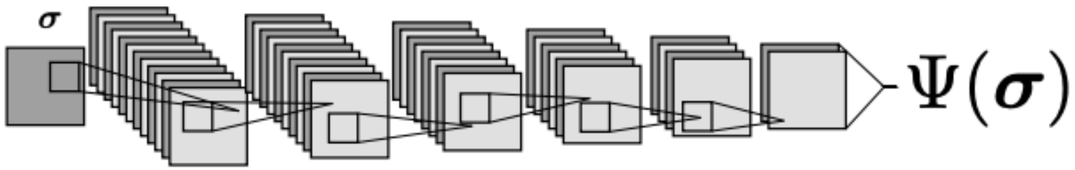
NetKet: https://arxiv.org/abs/1904.00031
https://www.netket.org/

ML for quantum problems

A. Neural-Network quantum states

Neural-network quantum states (NQS) are a representation of the many-body wave-function in terms of artificial neural networks (ANNs) (Carleo and Troyer, 2017). A commonly adopted choice is to parameterize wave-function amplitudes as a feed-forward neural network:

$$\Psi(\mathbf{r}) = g^{(L)}(W^{(L)} \dots g^{(2)}(W^{(2)} g^{(1)}(W^{(1)}\mathbf{r}))), \quad (3)$$




```python
import netket as nk

# Define the graph: a 1D chain of 20 sites with periodic
# boundary conditions
g = nk.graph.Hypercube(length=20, n_dim=1, pbc=True)

# Define the Hilbert Space: spin-half degree of freedom at each
# site of the graph, restricted to the zero magnetization sector
hi = nk.hilbert.Spin(s=0.5, total_sz=0.0, graph=g)

# Define the Hamiltonian: spin-half Heisenberg model
ha = nk.operator.Heisenberg(hilbert=hi)

# Define the ansatz: Restricted Boltzmann machine
# with 20 hidden units
ma = nk.machine.RbmSpin(hilbert=hi, n_hidden=20)

# Initialise with machine parameters
ma.init_random_parameters(seed=1234, sigma=0.01)

# Define the Sampler: metropolis sampler with local
# exchange moves, i.e. nearest neighbour spin swaps
# which preserve the total magnetization
sa = nk.sampler.MetropolisExchange(graph=g, machine=ma)

# Define the optimiser: Stochastic gradient descent with
# learning rate 0.01.
opt = nk.optimizer.Sgd(learning_rate=0.01)

# Define the VMC object: Stochastic Reconfiguration "Sr" is used
gs = nk.variational.Vmc(hamiltonian=ha, sampler=sa,
                        optimizer=opt, n_samples=1000,
                        use_iterative=True, method='Sr')

# Run the VMC simulation for 1000 iterations
# and save the output into files with prefix "test"
# The machine parameters are stored in "test.wf"
# while the measurements are stored in "test.log"
gs.run(output_prefix='test', n_iter=1000)
```

Example script for finding the ground state of the one-dimensional spin-$\frac{1}{2}$ Heisenberg model using an RBM ansatz.

Review: https://arxiv.org/abs/1903.10563

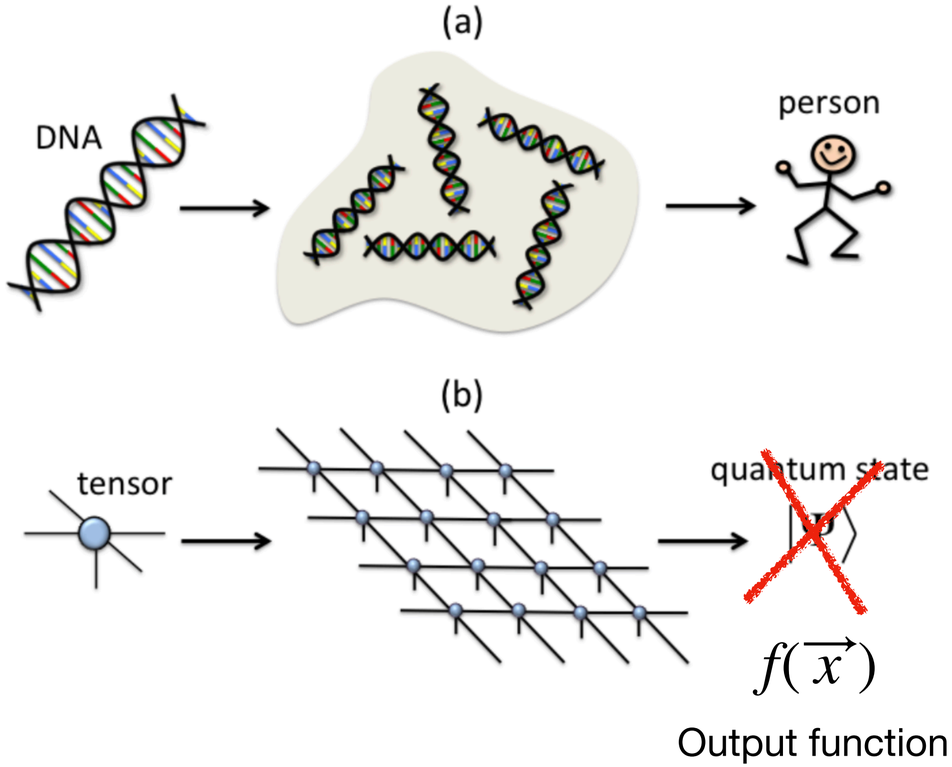
NetKet: https://arxiv.org/abs/1904.00031
https://www.netket.org/

Expressive power of NN quantum states: https://arxiv.org/abs/2103.10293

Quantum-inspired classical ML

Using knowledge of quantum to build a better classical ML models.

Tensor network to replace neural networks



Recommendation systems with superposition !

The main insight of this work is the use of simple routines to manipulate ℓ^2-norm sampling distributions, which play the role of quantum superpositions in the classical setting. This correspon-

- Incorporate **superposition in classical** manners
- Only **polynomially slower than quantum** version
- Exponential speed up from previous classical

[under some realistic conditions]

Further reading

Review on QML:

- **A non-review of Quantum Machine Learning: trends and explorations:** https://quantum-journal.org/views/qv-2020-03-17-32/

Quantum Computers:

- Quantum Computation and Quantum Information by Nielsen and Chuang
- **QC lectures by John Preskill**: https://www.youtube.com/playlist?list=PL0ojjrEqIyPy-1RRD8cTD_lF1hflo89Iu

Quantum-enhanced ML algorithms:

- Quantum Machine Learning: https://arxiv.org/abs/1611.09347

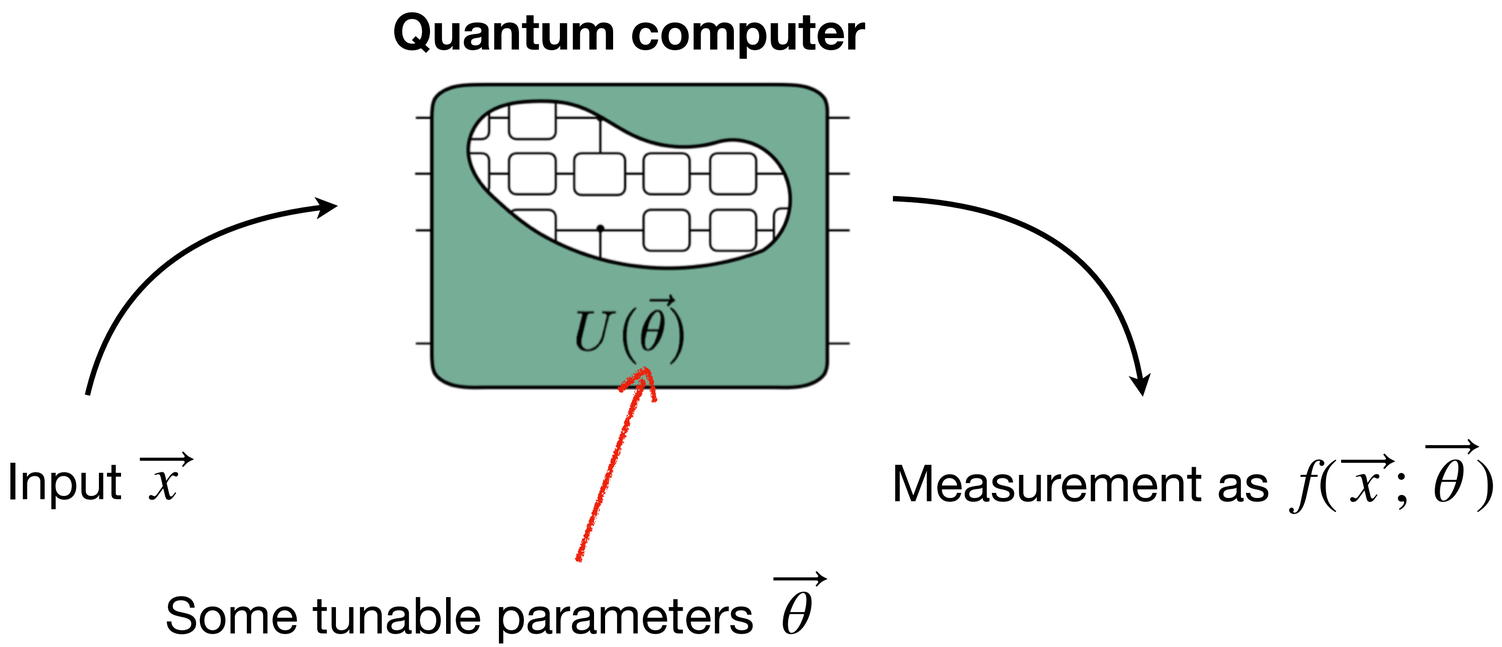
ML for quantum problems:

- Machine Learning and the physical sciences: https://arxiv.org/abs/1903.10563
- NetKet: https://arxiv.org/abs/1904.00031
  https://www.netket.org/
- Talk on Machine-Learning for Many-Body Quantum Physics: https://www.youtube.com/watch?v=4wBf-cOaC1M&ab_channel=CentreforQuantumTechnologies
- Expressive power of NN quantum states: https://arxiv.org/abs/2103.10293

Quantum-inspired classical ML:

- Supervised Learning with Quantum-Inspired Tensor Networks: https://arxiv.org/abs/1605.05775
- An overview of quantum-inspired classical sampling: https://ewintang.com/blog/2019/01/28/an-overview-of-quantum-inspired-sampling/
- A quantum-inspired classical algorithm for recommendation systems: https://arxiv.org/abs/1807.04271

Part 2: Quantum hardware as ML models



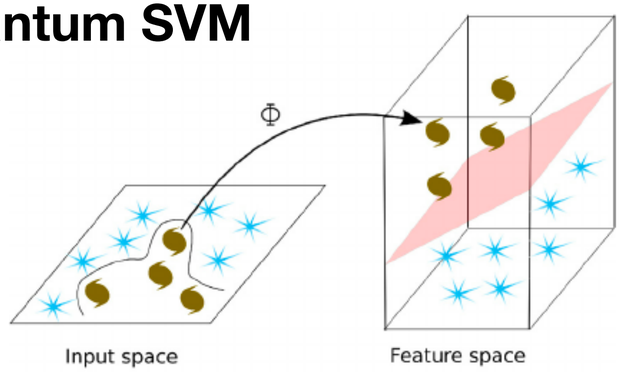
Recap: Quantum-enhanced ML

Translate existing algorithms into quantum version !

Focus on **less operations**

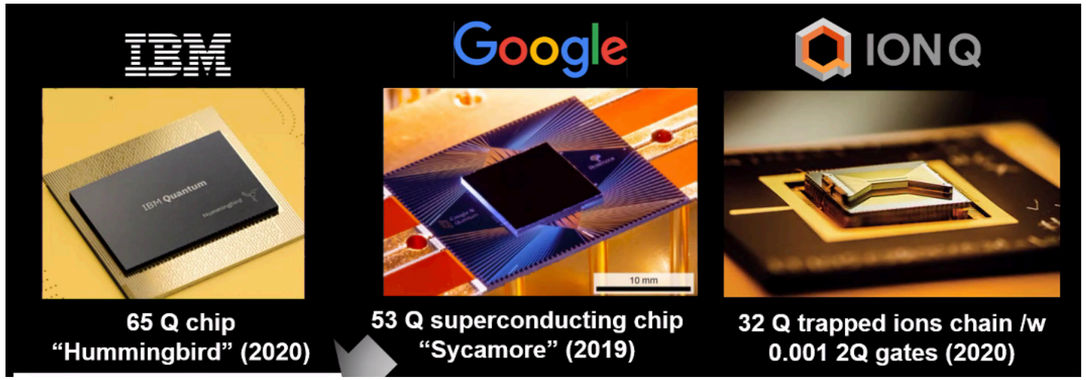
Focus on **more accurate results**

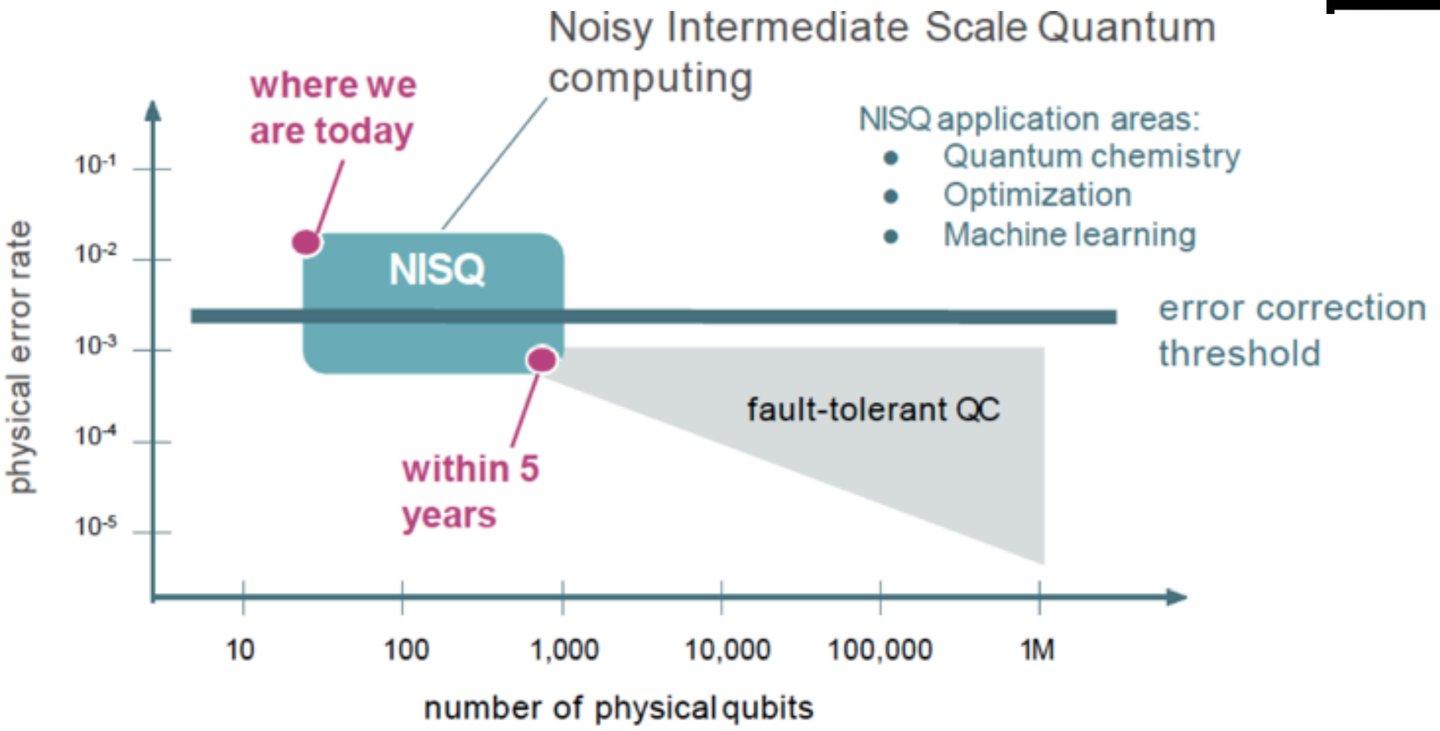
e.g. Quantum SVM



Noisy intermediate-scale quantum (NISQ) devices

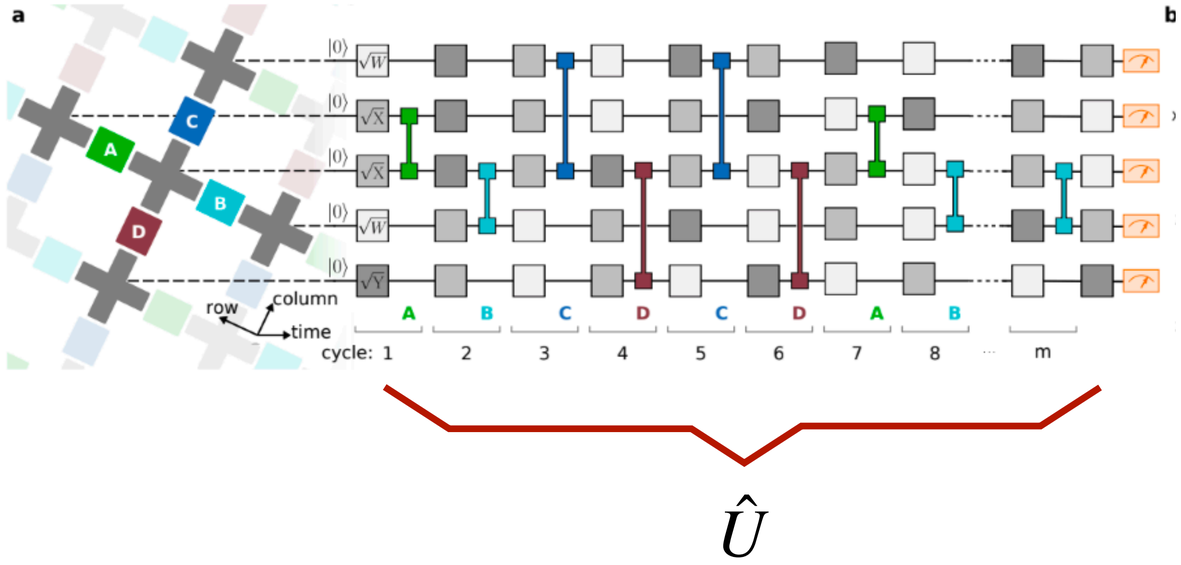
- 10 - 1000 physical qubits
- **No error correction**





Noisy intermediate-scale quantum (NISQ) devices

Already achieved **quantum supremacy**



$\hat{U}$

If random enough, simulate the whole Hilbert space of 2^n

nature

Explore content ▾ Journal information ▾ Publish with us ▾

nature › articles › article

Article | Published: 23 October 2019

Quantum supremacy using a programmable superconducting processor

Frank Arute, Kunal Arya, [...] John M

Nature **574**, 505–510(2019) | Cite t

817k Accesses | **849** Citations |

SHARE

REPORT

Quantum computational advantage using photons

Han-Sen Zhong[1,2,*], Hui Wang[1,2,*], Yu-Hao Deng[1,2,*], Ming-Cheng Chen[1,2,*], Li-Chao Peng[1,2], Yi-Han Luo[1,...]

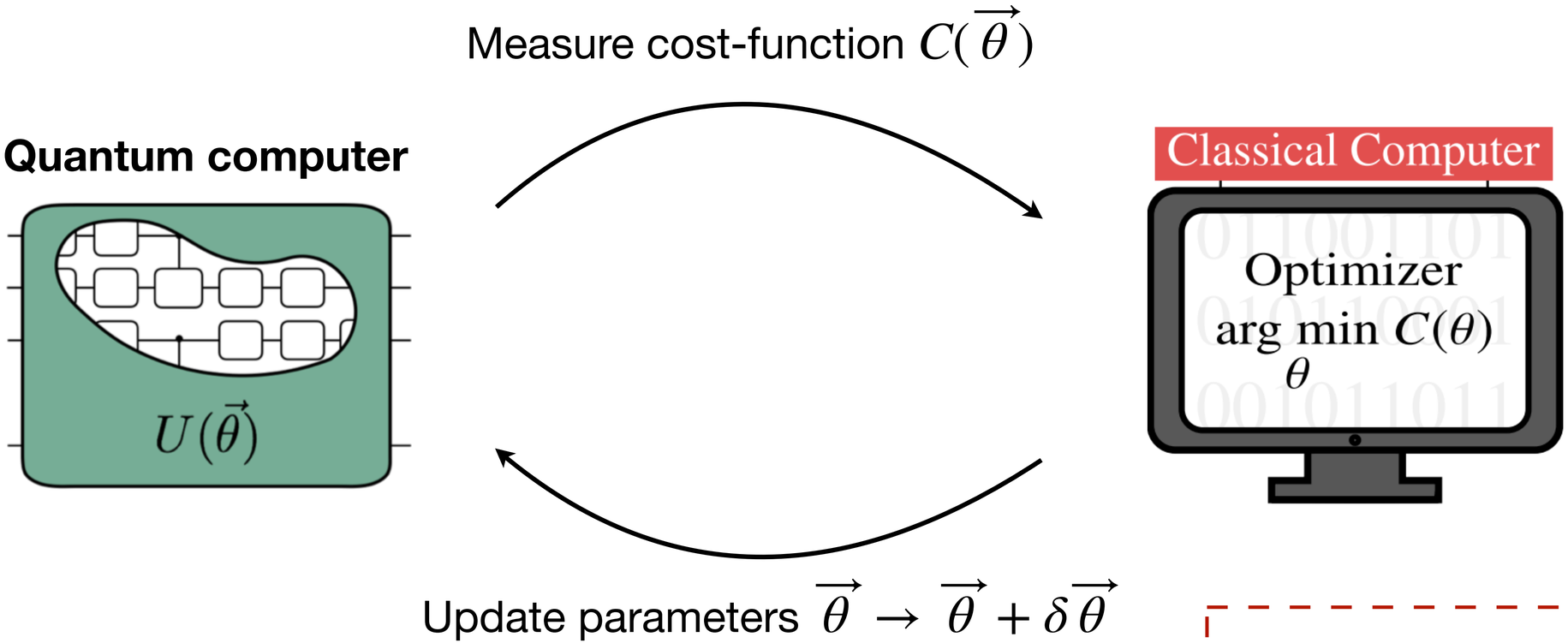
+ See all authors and affiliations

Science 18 Dec 2020:
Vol. 370, Issue 6523, pp. 1460-1463
DOI: 10.1126/science.abe8770

$n \sim 60$

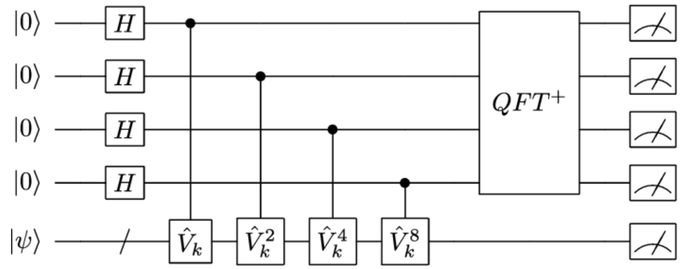
$2^{60} \sim 10^{18} >$ memory of supercomputers

Variational Quantum Algorithms (VQA)



- Cost-function defines a problem
- Shallow parameterised circuits $\hat{U}(\vec{\theta})$
- Feed-back loop mitigates some errors

Recap: traditional quantum algorithms



Examples of VQA

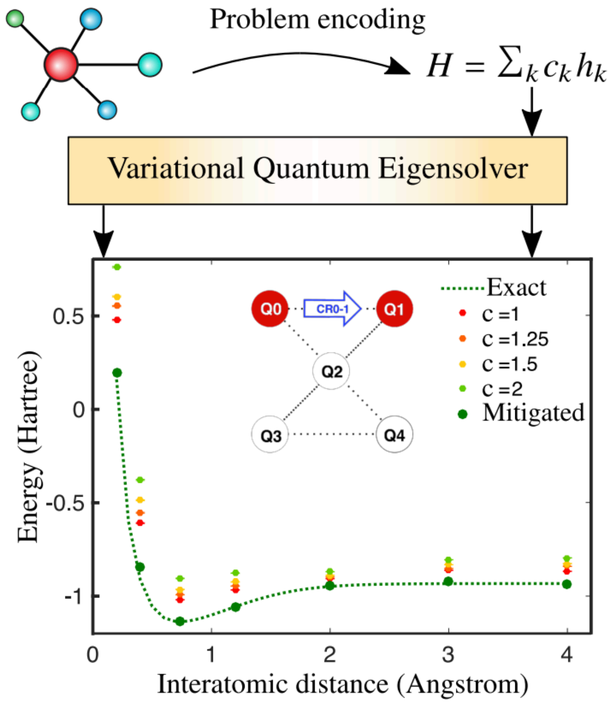
1. Finding ground-state energy of molecules

$$C(\vec{\theta}) = \langle \hat{H}(\vec{\theta}) \rangle$$

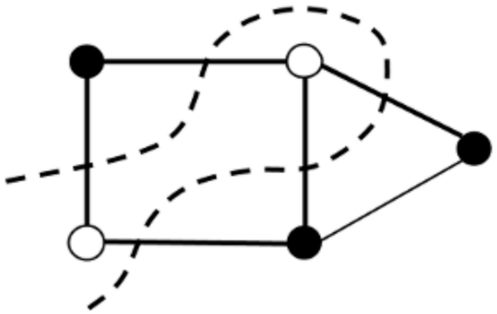
- Expectation of molecular Hamiltonian

$$\hat{H} = \sum_k c_k \hat{h}_k$$

- $\{\hat{h}_k\}$ are strings of Pauli matrices
  e.g. $\hat{Z}_1, \hat{Z}_2\hat{Z}_3, \hat{X}_1\hat{X}_2\hat{Y}_3\hat{Y}_4$



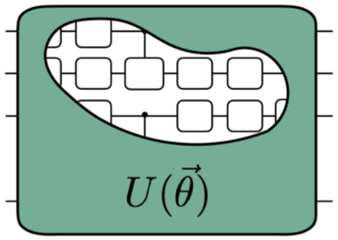
2. Combinatorial optimisation e.g. Max-Cut



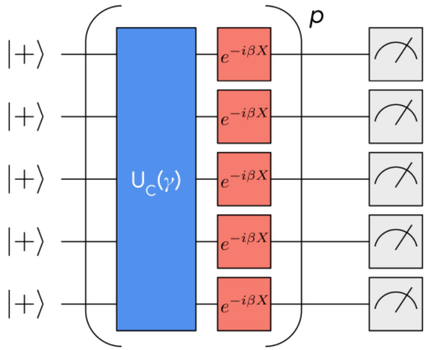
$$\hat{H}_{Ising} = \sum_{\langle i,j \rangle} A_{ij}(1 - \hat{Z}_i\hat{Z}_j)$$

$$C(\vec{\theta}) = \langle \hat{H}_{Ising}(\vec{\theta}) \rangle$$

- Expectation of Ising Hamiltonian



=



QAOA

Recap: Supervised learning

What is it ?

Given the training sample $\{\vec{x}_i, y_i\}_{i=1}^{N_s}$

Task is to approximate the underlying function $g(\vec{x})$ with the learning model at hand $f(\vec{x}; \vec{\theta})$

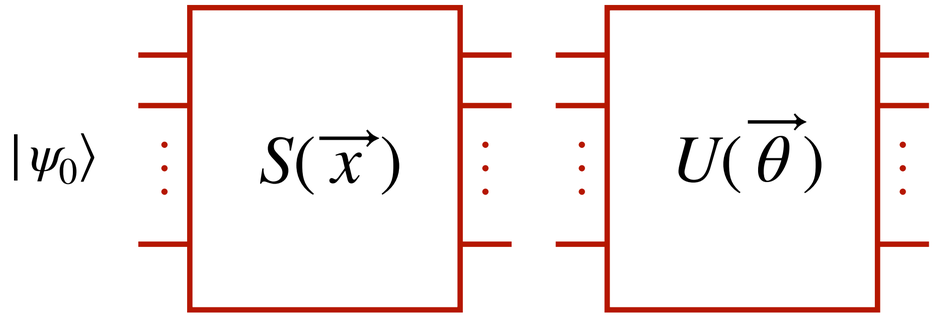
(such that it gives the correct labels on the unseen data)

How to achieve ?

Optimise the model's parameters $\vec{\theta}$ by minimising the cost function

$$\vec{\theta}_* = \text{argmin}_{\vec{\theta}} \sum_{i}^{N_s} C(y_i, f(\vec{x}_i; \vec{\theta}))$$

Quantum hardware as ML model



Data encoding block

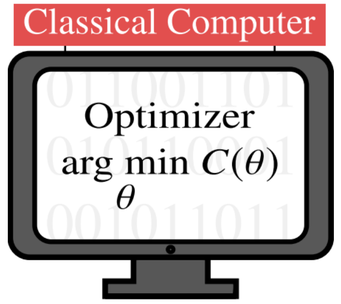
Training block

(often called **QNN**)

Expectation of an observable $\hat{O}$ as

$$f(\vec{x};\vec{\theta}) = \langle \hat{O}(\vec{x};\vec{\theta}) \rangle$$

Compute cost-function: $C(\vec{\theta}) = \sum_i \left(f(\vec{x}_i, \vec{\theta}) - y_i \right)^2$



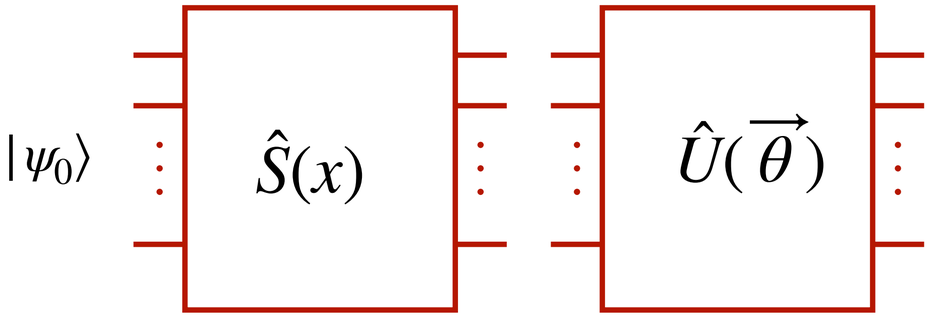
Update parameters

$$\vec{\theta} \rightarrow \vec{\theta} + \delta\vec{\theta}$$

What to investigate ?

- Expressibility of $f(x;\vec{\theta})$
- Trainability (how hard to optimise the model)
- Generalisation and possible quantum advantage

Expressibility of output function



Expectation of an observable $\hat{O}$ as

$$f(x;\vec{\theta}) = \langle \hat{O}(x;\vec{\theta}) \rangle$$

Assume $\hat{S}(x) = e^{ix\hat{H}_d}$ Encode x as evolution time of a **data encoding Hamiltonian** $\hat{H}_d$

$$f(x;\vec{\theta}) = \langle \hat{O}(x;\vec{\theta}) \rangle = \sum_{\omega\in\Omega} c_\omega(\vec{\theta}) e^{i\omega x}$$ (Fourier series type)

[i.e. universal approximator]

Frequencies depends on **data encoding Hamiltonian**

Expressibility of output function

$$f(x;\vec{\theta}) = \langle \hat{O}(x;\vec{\theta}) \rangle = \langle \psi | \hat{O} | \psi \rangle$$

Encoding: $S(x) = e^{ix\hat{H}_d} = \sum_j e^{ix\lambda_j} |\lambda_j\rangle\langle\lambda_j|$ with $\hat{H}_d |\lambda_j\rangle = \lambda_j |\lambda_j\rangle$

$$|\psi\rangle = \hat{U}(\vec{\theta})\hat{S}(x)|\psi_0\rangle = \hat{U}(\vec{\theta})\left(\sum_j e^{ix\lambda_j}|\lambda_j\rangle\langle\lambda_j|\right)|\psi_0\rangle = \sum_j e^{ix\lambda_j}\hat{U}(\vec{\theta})|\lambda_j\rangle\langle\lambda_j|\psi_0\rangle$$

$$\langle\psi| = \sum_k e^{-ix\lambda_k}\langle\psi_0|\lambda_k\rangle\langle\lambda_k|\hat{U}^\dagger(\vec{\theta})$$

$$f(x;\vec{\theta}) = \langle\psi|\hat{O}|\psi\rangle = \left(\sum_k e^{-ix\lambda_k}\langle\psi_0|\lambda_k\rangle\langle\lambda_k|\hat{U}^\dagger(\vec{\theta})\right)\hat{O}\left(\sum_j e^{ix\lambda_j}\hat{U}(\vec{\theta})|\lambda_j\rangle\langle\lambda_j|\psi_0\rangle\right)$$

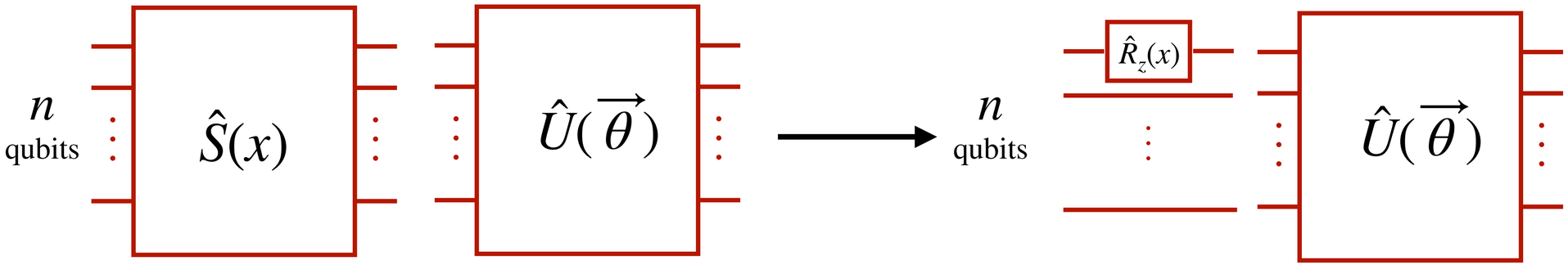
$$= \sum_{k,j} e^{ix(\lambda_j-\lambda_k)}\left(\langle\psi_0|\lambda_k\rangle\langle\lambda_k|\hat{U}^\dagger(\vec{\theta})\hat{O}\hat{U}(\vec{\theta})|\lambda_j\rangle\langle\lambda_j|\psi_0\rangle\right)$$

$$= \sum_{\omega\in\Omega} c_\omega(\vec{\theta})e^{i\omega x}$$

$$c^*_\omega(\vec{\theta}) = c_{-\omega}(\vec{\theta})$$

$$\omega = \lambda_j - \lambda_k \qquad (j,k \in \{1,\ldots,2^n\})$$

Example: single-quit Pauli rotation



$$\hat{R}_z(x) = e^{\frac{ix}{2}\hat{Z}}$$

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

Unique eigenvalues: $\lambda_j = -1/2, +1/2$

Frequency spectrum: $\Omega = \{\lambda_j - \lambda_k\}$

$= -1, 0, 1$

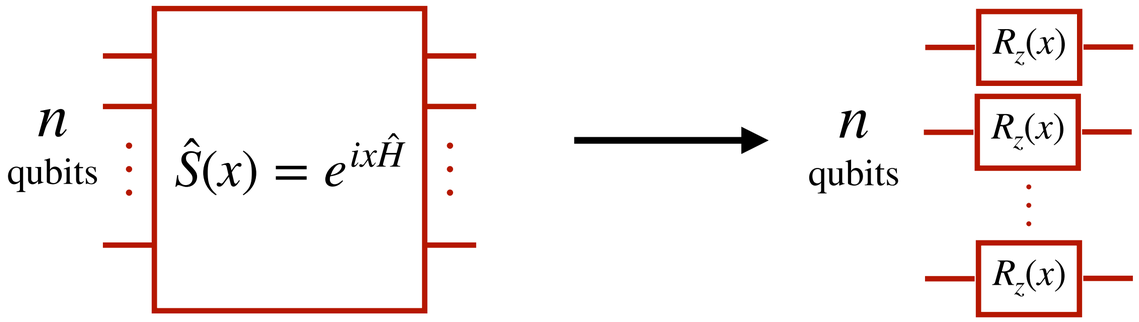
$$f(x; \vec{\theta}) = \sum_{\omega \in \Omega} c_\omega(\vec{\theta}) e^{i\omega x} = c_{-1}e^{-ix} + c_0 + c_1 e^{ix}$$

$$= c_0 + A \sin x + B \cos x$$

$$c_{-1} = c_1^*$$

Regardless of how complex $\hat{U}(\vec{\theta})$

Example: single-quit Pauli rotations in parallel



$$\hat{S}(x) = e^{\frac{ix}{2}\hat{Z}} \otimes e^{\frac{ix}{2}\hat{Z}} \otimes \ldots \otimes e^{\frac{ix}{2}\hat{Z}}$$

$$\hat{H} = \frac{1}{2}\sum_{i}^{n} \hat{Z}_i$$

Unique eigenvalues

$$\lambda_p = \left(\pm\frac{1}{2} \pm \frac{1}{2} \pm \ldots \pm \frac{1}{2} \right)$$

$$\lambda_p = \left(\frac{p}{2} - \frac{n-p}{2} \right) \quad , p \in \{0,\ldots,n\}$$

↙ # of $+1/2$ ↘ # of $-1/2$

Frequency spectrum

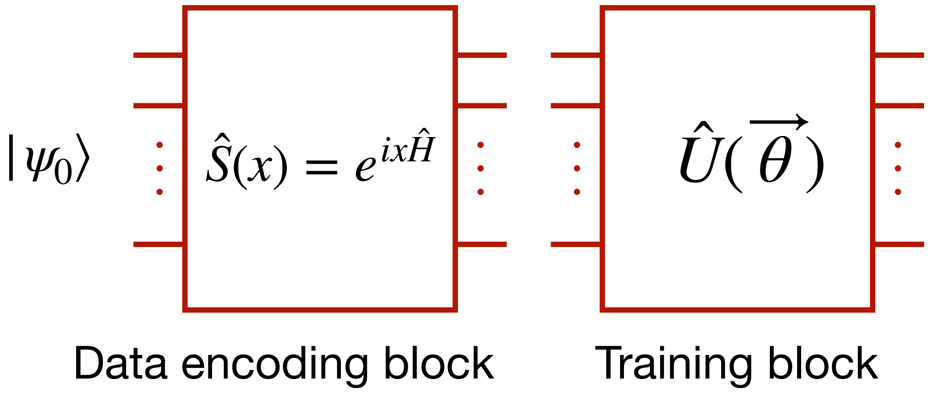
$$\Omega_{R_z} = \{\lambda_p - \lambda_{p'}\}$$

$$= \{p - p' \,|\, p, p' \in \{0,\ldots,n\}\}$$

$$= \{-n, -(n-1), \ldots, 0, \ldots, (n-1), n\}$$

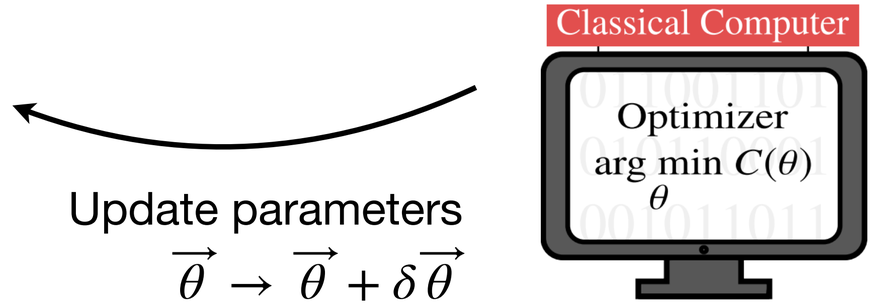
$$f(x) = c_0 + \sum_{k=1}^{n} A_k \sin(kx) + \sum_{k=1}^{n} B_k \cos(kx)$$

Trainability

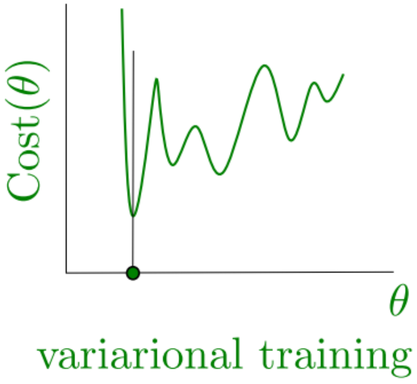


$$f(x;\vec{\theta}) = \langle \hat{O}(x;\vec{\theta}) \rangle = \sum_{\omega \in \Omega} c_\omega(\vec{\theta}) e^{i\omega x}$$

depends on $\hat{H}_d$



- **Expressibility of** $f(x;\vec{\theta})$ depends on **how we encode data** into our quantum model.
- Form of $\hat{U}(\vec{\theta})$ greatly tells **trainability of the model i.e. how easy to control** c_ω



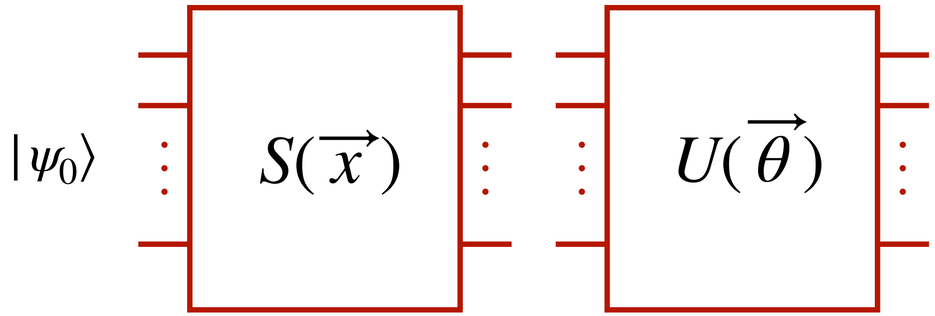
Two extreme examples:

$\hat{U}(\vec{\theta}) = \hat{\mathbb{1}}$ ⟶ Cannot train

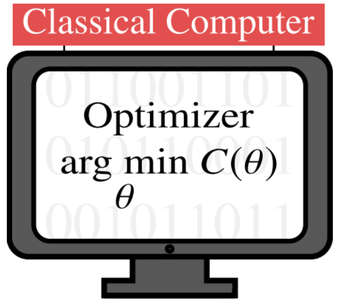
$\hat{U}(\vec{\theta}) =$ any $\hat{U}$ ⟶ Also **cannot train !!**

Trainability (Barren Plateaus)

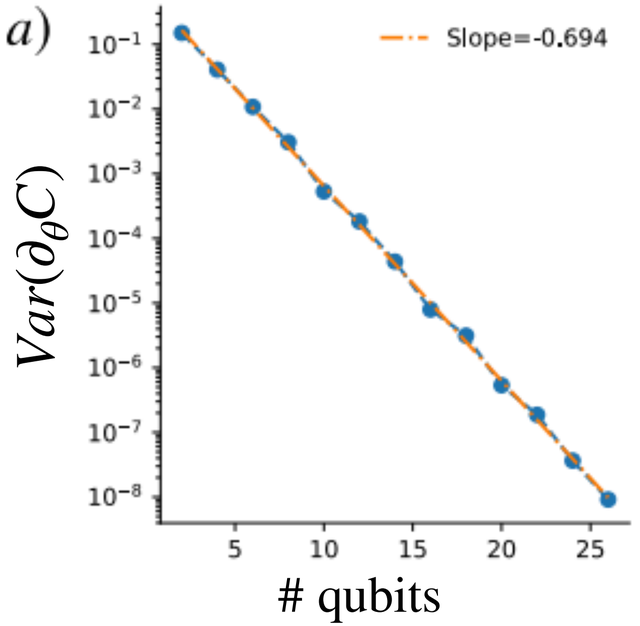
$\hat{U}(\vec{\theta}) = \text{any } \hat{U}$ ⟶ Also **cannot train !!**

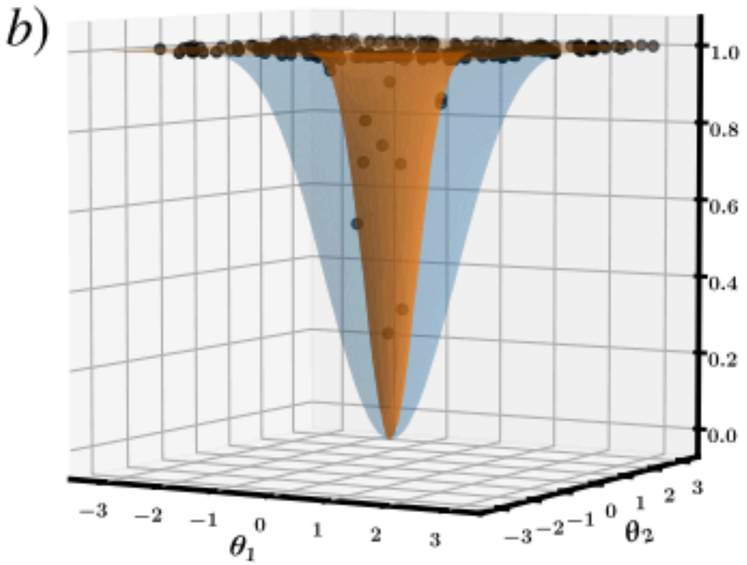


Compute cost-function: $C(\vec{\theta}) = \sum_i \left(f(\vec{x}_i, \vec{\theta}) - y_i\right)^2$



Update parameters
$\vec{\theta} \to \vec{\theta} + \delta\vec{\theta}$





$\hat{U}(\vec{\theta}) = \text{any } \hat{U}$

exhibit **chaoticity**

↓

Exponentially flat landscape of $C(\vec{\theta})$

Further reading (II)

Reviews on NISQ era

- **Variational Quantum Algorithms**: https://arxiv.org/abs/2012.09265
- **Noisy intermediate-scale quantum (NISQ) algorithms:** https://arxiv.org/abs/2101.08448
- Quantum Computing in the NISQ era and beyond: https://quantum-journal.org/papers/q-2018-08-06-79/

Review on Quantum Supremacy

- Quantum supremacy blog by google: https://ai.googleblog.com/2019/10/quantum-supremacy-using-programmable.html

Expressibility of output function

- **The effect of data encoding on the expressive power of variational quantum machine learning models:** https://arxiv.org/abs/2008.08605

Barren Plateaus

- Barren plateaus in quantum neural network training landscapes: https://www.nature.com/articles/s41467-018-07090-4
- Connecting ansatz expressibility to gradient magnitudes and barren plateaus: https://arxiv.org/abs/2101.02138
- Noise-Induced Barren Plateaus in Variational Quantum Algorithms: https://arxiv.org/abs/2007.14384

Softwares for Quantum Computers

- Qiskit by IBM: https://qiskit.org/
- Tensorflow quantum by Google: https://www.tensorflow.org/quantum
- Pennylane by Xanadu: https://pennylane.ai/

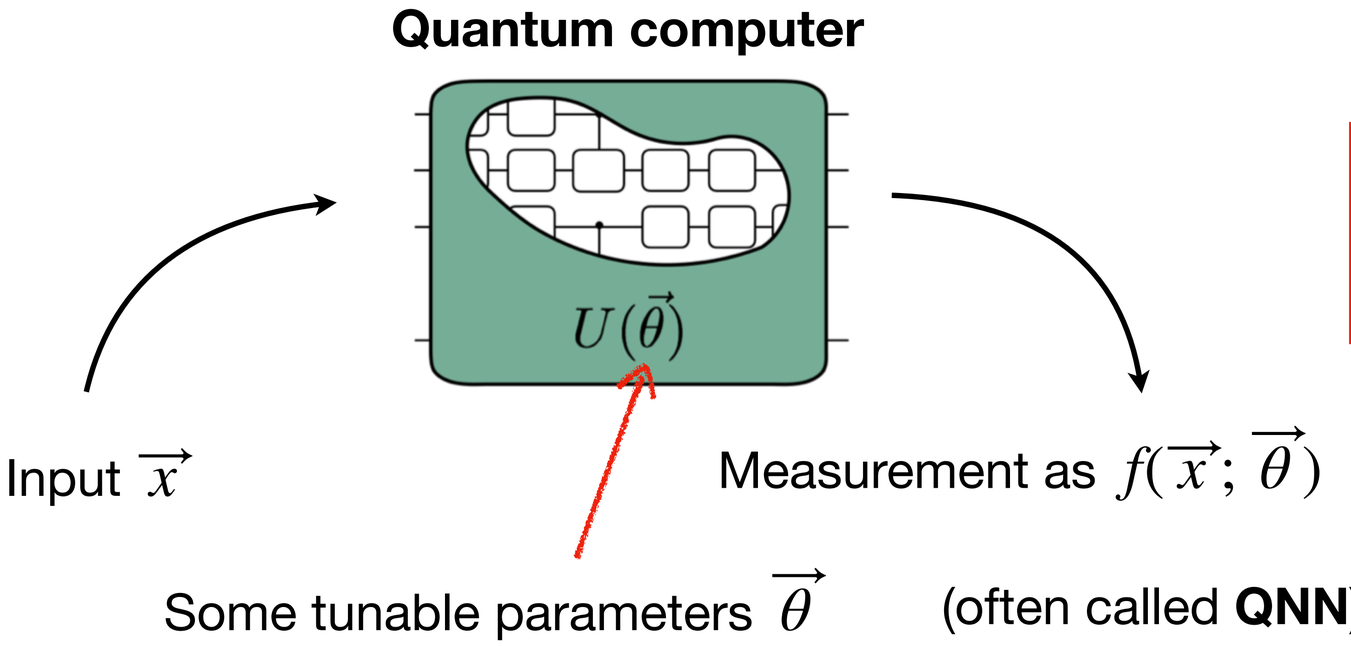
Part 3: Quantum Kernels and possible advantage

Quantum Kernel

- **Supervised quantum machine learning models are kernel methods**: https://arxiv.org/abs/2101.11020

Possible quantum advantage

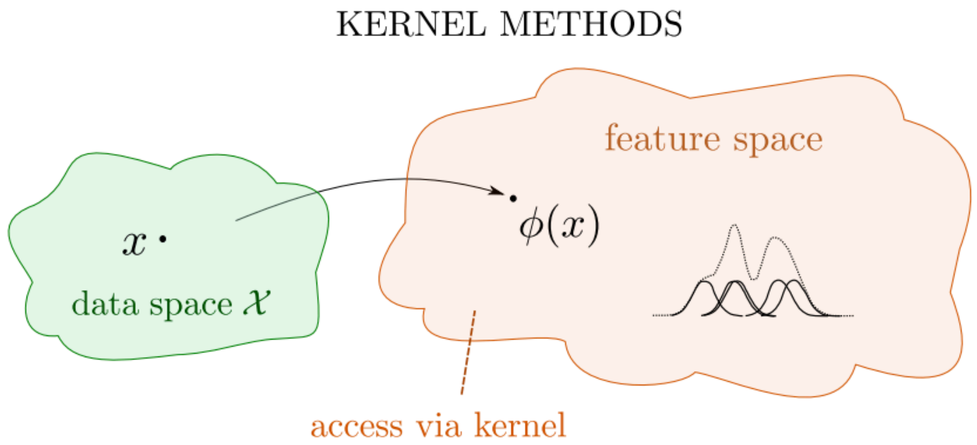
- **Power of data in quantum machine learning**: https://arxiv.org/abs/2011.01938
- Talk on this paper: https://www.youtube.com/watch?v=0SrrftaEiD8&t=1630s&ab_channel=CentreforQuantumTechnologies

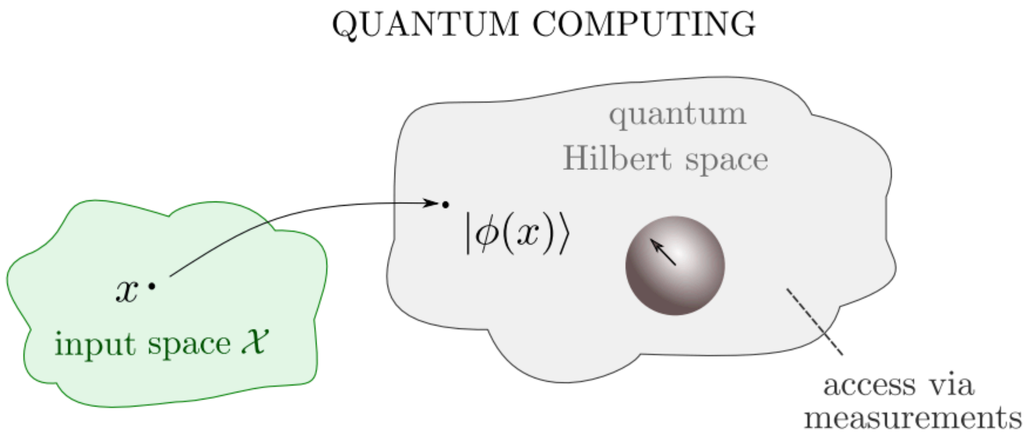


Focus on **more accurate results**

Quantum Kernel

- Notice **similarities** between **quantum model** and **classical kernel methods.**



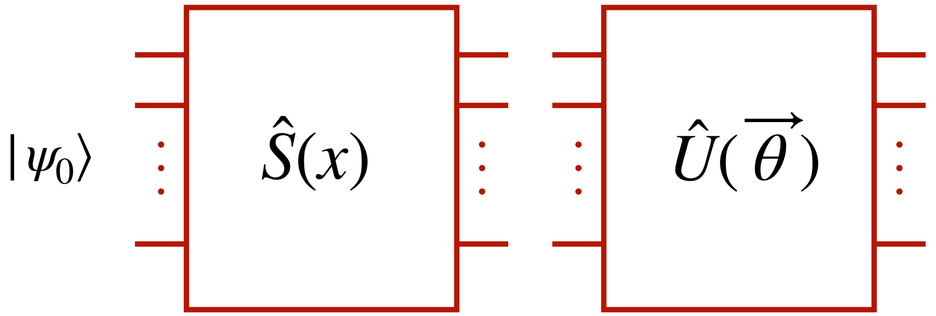


$$k_Q(x, x') = |\langle \phi(x) | \phi(x') \rangle|^2$$ **(Quantum kernel)**

Quantum models embedded with data can fundamentally be formulated as a **classical kernel method** whose **kernel is computed by a quantum computer.**

[i.e. it is naturally more similar to kernel **rather than neural network !!**]

Quantum kernels: $k_Q(x,x') = |\langle\phi(x)|\phi(x')\rangle|^2 = Tr[\rho(x)\rho(x')]$ with $\rho(x) = |\phi(x)\rangle\langle\phi(x)|$



Data encoding block Training block

Expectation of an observable $\hat{O}$ as

$$f(x;\vec{\theta}) = \langle\hat{O}(x;\vec{\theta})\rangle$$
$$= \langle\phi(x)|\hat{U}^\dagger(\vec{\theta})\hat{O}\hat{U}(\vec{\theta})|\phi(x)\rangle$$

with $|\phi(x)\rangle = \hat{S}(x)|\psi_0\rangle$

$$f(x;\vec{\theta}) = Tr[(\hat{U}^\dagger(\vec{\theta})\hat{O}\hat{U}(\vec{\theta}))|\phi(x)\rangle\langle\phi(x)|] = Tr[\hat{M}(\vec{\theta})\rho(x)]$$ with $\hat{M}(\vec{\theta}) = \hat{U}^\dagger(\vec{\theta})\hat{O}\hat{U}(\vec{\theta})$

Representer Theorem (from kernel's methods)

Optimal model: $$f_{opt}(x) = \sum_i^{N_s} \alpha_i^{opt} Tr[\rho(x_i)\rho(x)] = Tr\left[\overbrace{\left(\sum_i^{N_s}\alpha_i^{opt}\rho(x_i)\right)}^{\hat{M}_{opt}}\rho(x)\right]$$

Alternative way to train ?

$$f_{opt}(x) = \sum_i^{N_s} \alpha_i^{opt} Tr[\rho(x_i)\rho(x)] = Tr\left[\left(\sum_i^{N_s} \alpha_i^{opt}\rho(x_i)\right)\rho(x)\right]$$

$$\hat{M}_{opt}$$

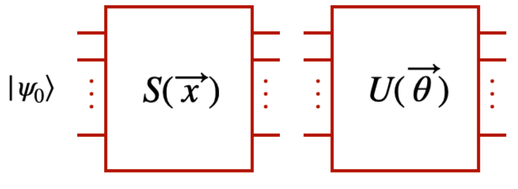
Kernel-based training (this is simply regression !)

$$f(x) = \sum_i^{N_s} \alpha_i Tr[\rho(x_i)\rho(x)]$$

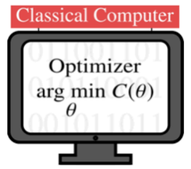
Parameters

from quantum computers

Variational training

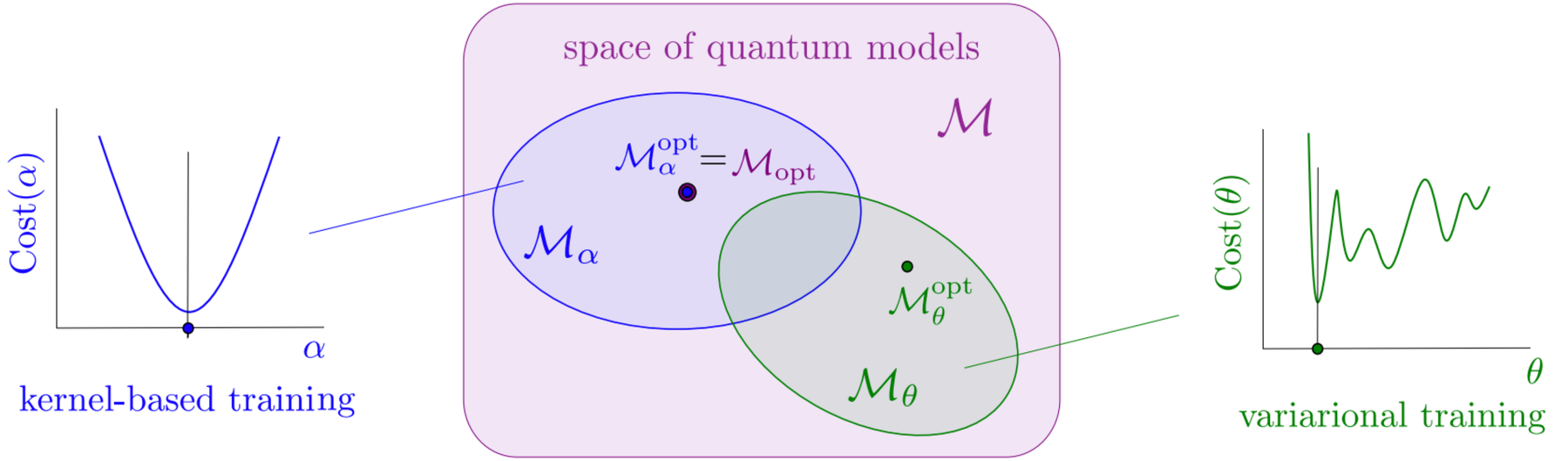


Compute cost-function: $C(\vec{\theta}) = \sum_i \left(f(\vec{x}_i, \vec{\theta}) - y_i\right)^2$



Update parameters
$\vec{\theta} \to \vec{\theta} + \delta\vec{\theta}$

- Guarantee the best model given the training sample
- Avoid Barren plateaus



Challenges in training ?

Kernel-based training (this is simply regression !)

$$f(x) = \sum_{i}^{N_s} \alpha_i Tr[\rho(x_i)\rho(x)]$$

Parameters — from quantum computers

- Guarantee the best model given the training sample
- Avoid Barren plateaus

During training:

- Evaluate **kernels of all possible pairs of training data !**

$$K_Q = [k_Q(x_i, x_j)] = \left[Tr(\rho(x_i)\rho(x_j))\right]_{i,j=1}^{N_x}$$

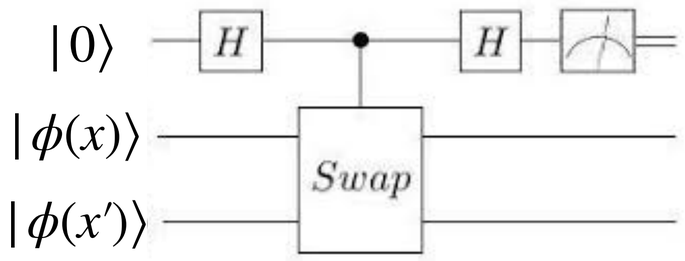
After training:

- Evaluate **kernels of new data point x_{new} with all training data !**

$$f(x_{new}) = \sum_{i}^{N_s} \alpha_i^{opt} Tr[\rho(x_i)\rho(x_{new})]$$

How to efficiently compute kernels from Quantum Computers ? [i.e. not so sure how with NISQ devices

$$k_Q(x, x') = Tr(\rho(x)\rho(x')) = |\langle\phi(x)|\phi(x')\rangle|^2$$



$$\langle\hat{Z}\rangle = |\langle\phi(x)|\phi(x')\rangle|^2$$

SWAP test

Possible Quantum Advantage

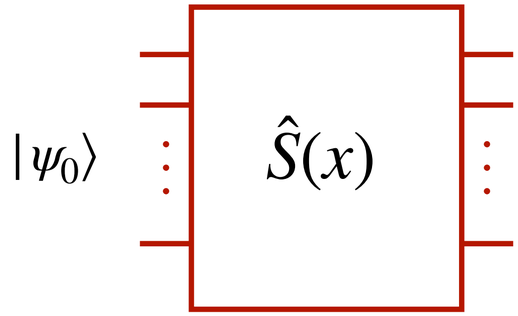
- Assume we can train efficiently

Focus on **more accurate results**
(than classical ML models)

complexity/accuracy $f_{opt}(x) = \sum_{i}^{N_s} \alpha_i^{opt} Tr[\rho(x_i)\rho(x)]$

[Implicitly, $k(x, x')$ cannot be efficiently computed classically]

Quantum kernels [How we **encode data into quantum devices**]



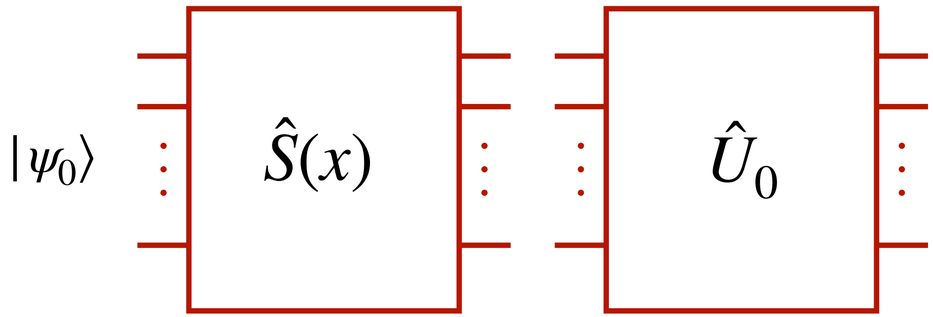
Data encoding block

$$k_Q(x, x') = Tr[\rho(x)\rho(x')]$$
$$= |\langle \phi(x) | \phi(x') \rangle|^2$$

with $|\phi(x)\rangle = \hat{S}(x)|\psi_0\rangle$

Power of data in classical ML

Task: Classical ML model to **learn a target quantum ML model** $g(x)$



$$g(x) = \langle \phi(x) | \hat{U}_0^\dagger \hat{O} \hat{U}_0 | \phi(x) \rangle$$

A motivational example: consider the amplitude encoding for $x \in R^N$ $\quad |\phi(x)\rangle = \hat{S}(x)|\psi_0\rangle = \sum_{i=1}^{N} x^{(i)} |i\rangle$

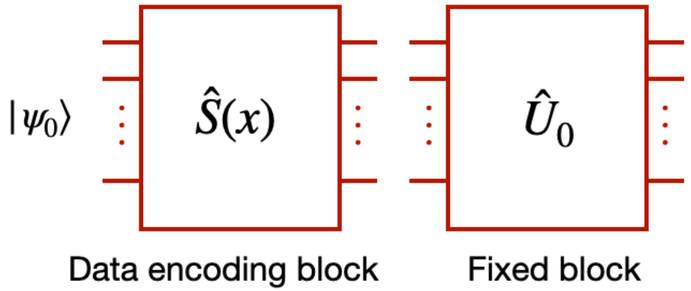
$$g(x) = \langle \phi(x) | \hat{U}_0^\dagger \hat{O} \hat{U}_0 | \phi(x) \rangle = \sum_{i,j} B_{ij} x^{(i)*} x^{(j)} \quad \text{with} \quad B_{ij} = \langle i | \hat{U}_0^\dagger \hat{O} \hat{U}_0 | j \rangle$$

Without training data, simulating $g(x)$ **cannot be done efficeintly with classical** computers in general

With training data $\{x_p, y_p\}_{p=1}^{N_s}$**,** we can train the classical model i.e. this is **just quadric fitting**.

Prediction bound by kernel-based models

Task: Kernel-based ML model to **learn a target quantum ML model $g(x)$, given a training set**



$$g(x) = \langle \phi(x) | \hat{U}_0^\dagger \hat{O} \hat{U}_0 | \phi(x) \rangle$$

$$\{x_p, y_p = g(x_p)\}_{p=1}^{N_s}$$

[now arbitrary encoding]

Prediction error bound: [given a kernel $K_{ij} = k(x_i, x_j)$ after training]

$$\mathop{\mathbb{E}}_{x\sim D} |f(x) - g(x)| \leq c\sqrt{\frac{s_K}{N_s}} \qquad \text{with} \qquad s_K = \sum_{i,j=1}^{N_s} (K_{ij}^{-1}) y_i y_j$$

Quantum kernel: $K_{ij}^Q = k_Q(x_i, x_j) = Tr(\rho(x_i)\rho(x_j))$

$$\mathop{\mathbb{E}}_{x\sim D} |f_Q(x) - g(x)| \leq c\sqrt{\frac{\min(d, Tr(\hat{O}^2))}{N_s}} \qquad \text{with} \qquad d = \dim(\hat{P}_Q) = \text{rank}(K^Q) \leq N_s$$

Projector onto subspace by $\{\text{vec}(\rho(x_1)), \ldots, \text{vec}(\rho(x_{N_s}))\}$

[depends only on input data $\{x_i\}$]

Comparison between two kernel-based models

Given two trained models with kernels K^C and K^Q

$$s_C \leq g_{CQ}^2 s_Q$$

with

Geometric difference

$$g_{CQ} = g(K^C \| K^Q) = \sqrt{\| \sqrt{K^Q} (K^C)^{-1} \sqrt{K^Q} \|_\infty}$$

[depends only on input data $\{x_i\}$]

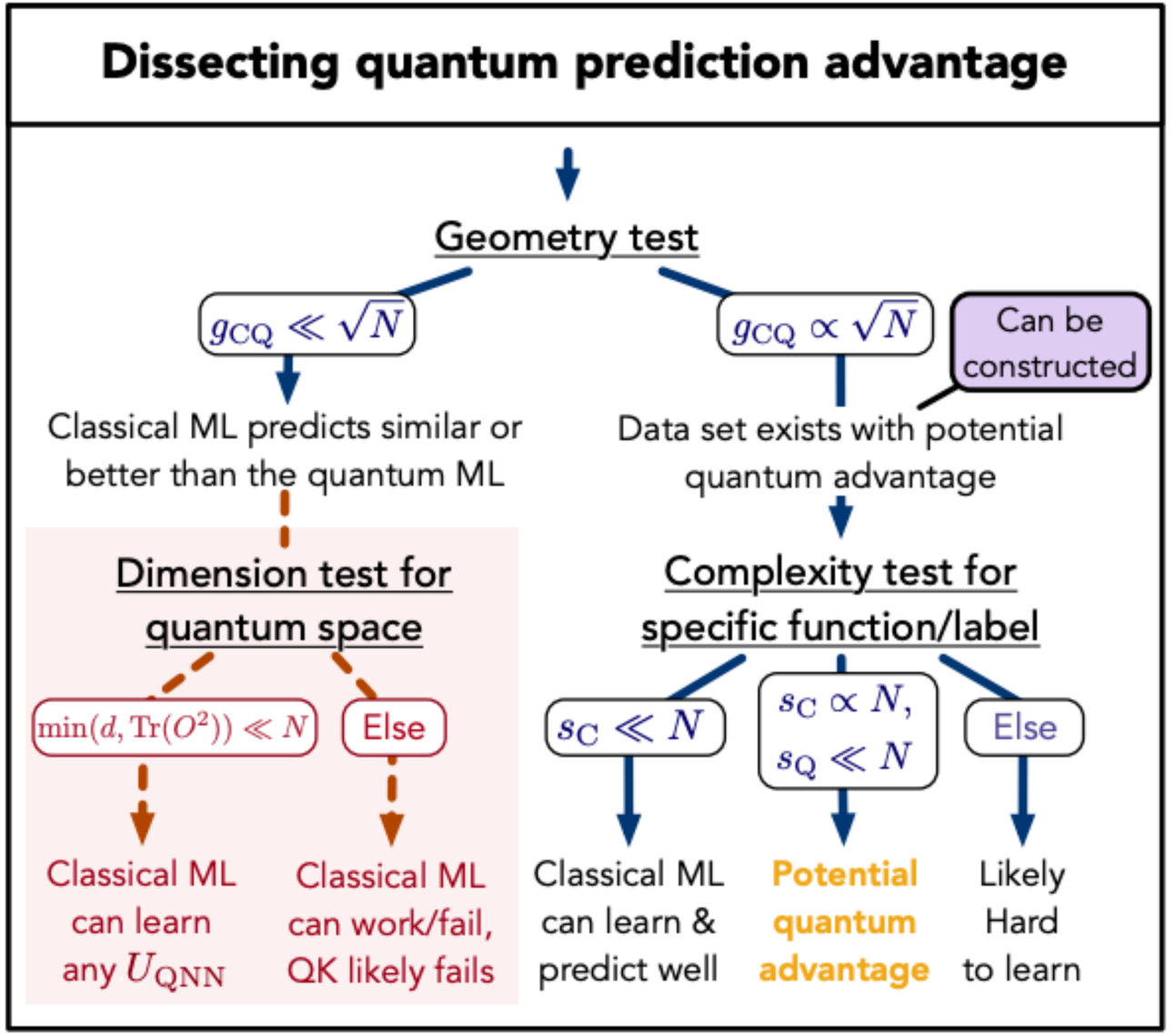
Small g_{CQ} ⟶ Classical similar/better than Quantum

Large g_{CQ} ⟶ Quantum may be able to win

[In the paper, they can engineer data set such that

$s_C = g_{CQ}^2 s_Q$ i.e. Quantum advantage]

Comparison between two kernel-based models



Further reading (III)

Quantum Kernel

- **Supervised quantum machine learning models are kernel methods**: https://arxiv.org/abs/2101.11020

Possible quantum advantage

- **Power of data in quantum machine learning**: https://arxiv.org/abs/2011.01938
- Talk on this paper: https://www.youtube.com/watch?v=0SrrftaEiD8&t=1630s&ab_channel=CentreforQuantumTechnologies

Thank you and Q&A

Further Discussion

Given two trained models with kernels K^C and K^Q

$$s_C \leq g_{CQ}^2 s_Q$$

with

Geometric difference

$$g_{CQ} = g(K^C \| K^Q) = \sqrt{\| \sqrt{K^Q} (K^C)^{-1} \sqrt{K^Q} \|_\infty}$$

[depends only on input data $\{x_i\}$]

When the quantum states $\rho(x^i)$ for the training set span a large dimension quantum Hilbert space, all inputs are too far apart, so

$$K^Q \approx I \quad \text{and} \quad g_{CQ} = \sqrt{\| \sqrt{K_Q} K_C^{-1} \sqrt{K_Q} \|_\infty} \approx 1.$$

This means classical ML can often compete or outperform quantum kernel methods in learning any quantum models.

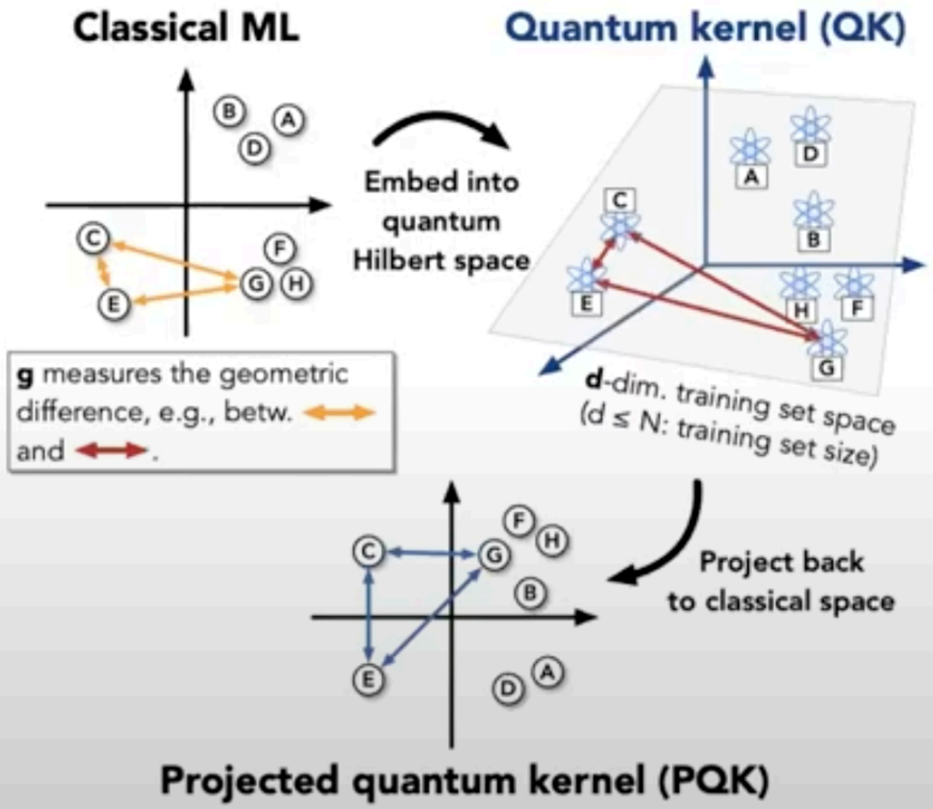
Further Discussion

Large quantum Hilbert space dimension makes quantum ML suffers more than classical ML.

Projects quantum states back to classical space, e.g. using reduced observable or classical shadow [1].

Define kernel in the classical space.

We call this the projected quantum kernel (PQK).



[1] Predicting many properties of a quantum system from very few measurements