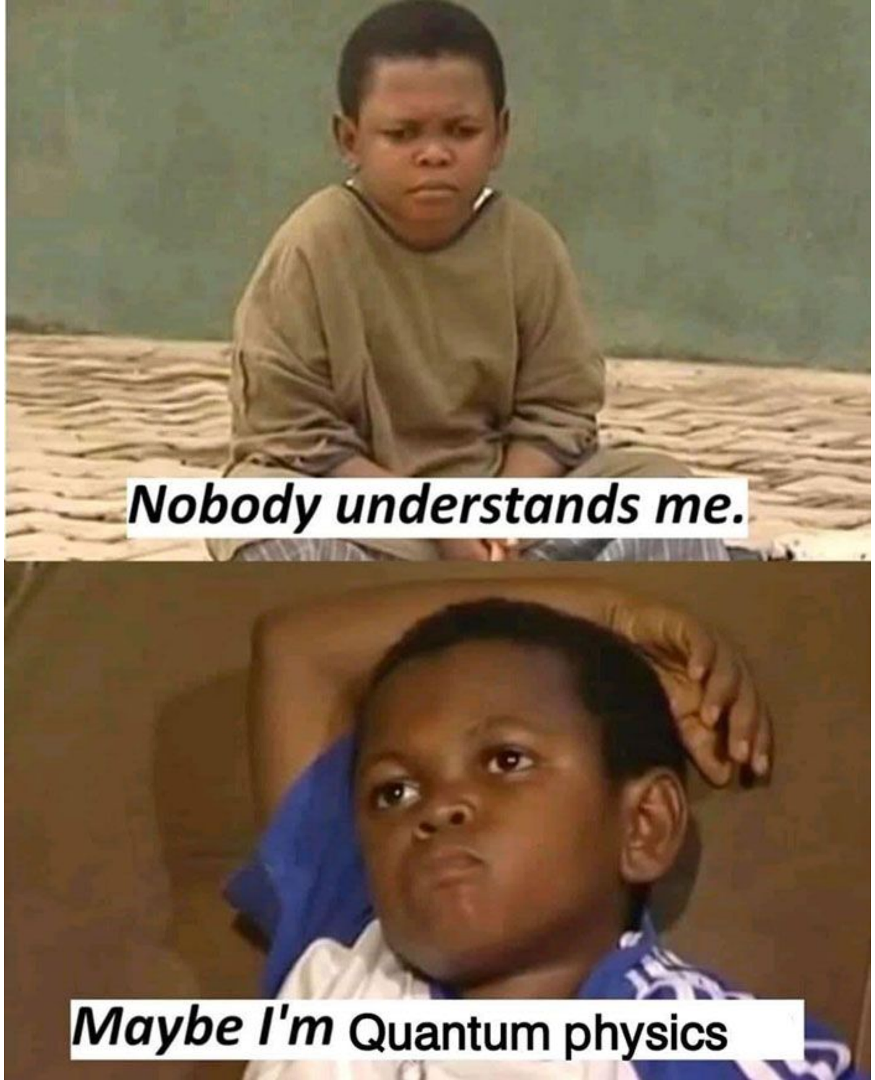
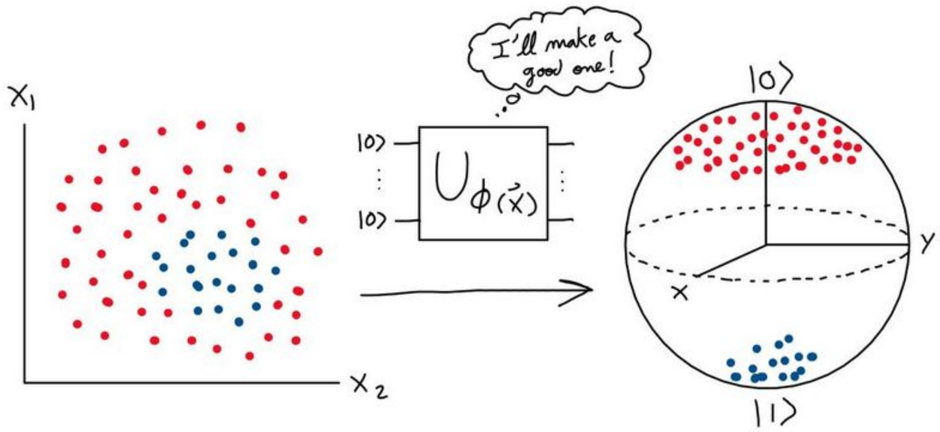
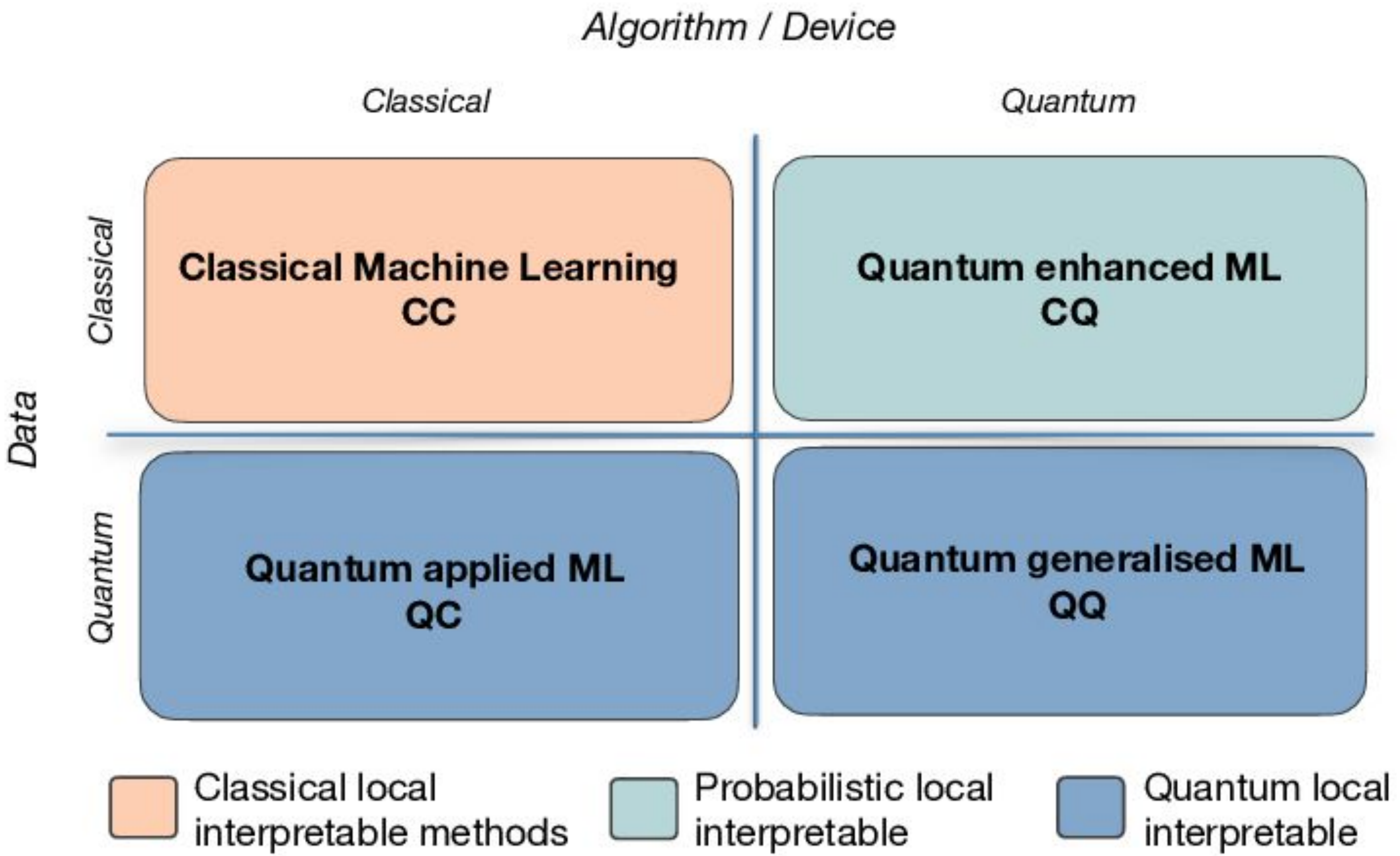
This is CS4084!

GCR:wzj3vua


Nobody understands me.
Maybe I'm Quantum physics

Quantum Embedding




Algorithm / Device
Classical
Quantum
Data
Classical
Quantum
Classical Machine Learning
CC
Quantum enhanced ML
CQ
Quantum applied ML
QC
Quantum generalised ML
QQ
Classical local interpretable methods
Probabilistic local interpretable
Quantum local interpretable

$$2^2 = 4\ states \longrightarrow \left.\begin{array}{l} \xrightarrow{\text{State 1}} 0\,0 \\ \xrightarrow{\text{State 2}} 0\,1 \\ \xrightarrow{\text{State 3}} 1\,0 \\ \xrightarrow{\text{State 4}} 1\,1 \end{array}\right\} \text{Classical Computers process only one state at a time}$$

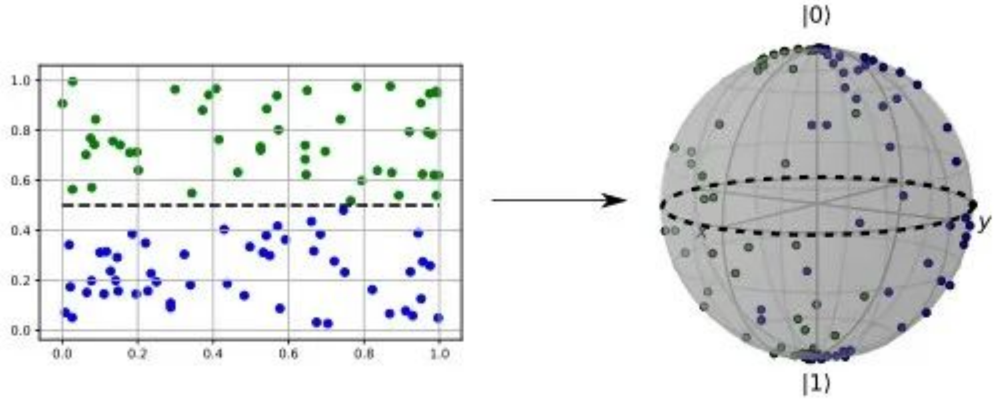
$$\overbrace{\alpha|00\rangle+\beta|01\rangle+\gamma|10\rangle+\delta|11\rangle}^{\text{Quantum Computers process all states at a time based on amplitudes or probabilities}}$$

Here, the amplitudes $\alpha,\beta,\gamma,\delta$ *are complex numbers and the sum of the squares of the absolute values is 1; i.e.,* $|\alpha|^2+|\beta|^2+|\gamma|^2+|\delta|^2 = 1.$

Quantum Data

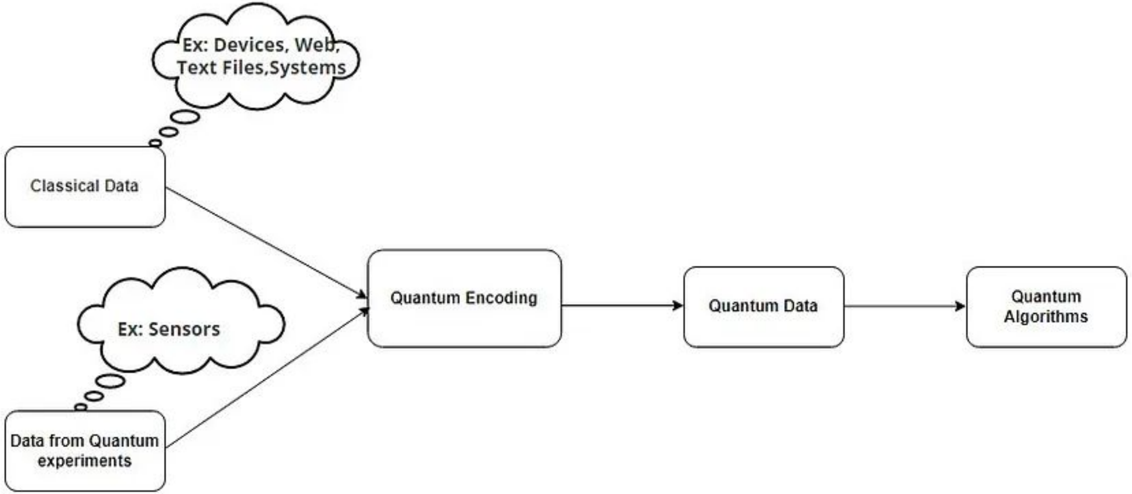
Working directly with classical data is not possible and inefficient as well in Quantum Computing.

Quantum Computers has to be compatible with data and data has to be in the form of Quantum Laws like Superposition and Entanglement.



Quantum Data

To load the data in quantum computer, it must be encoded in quantum bits (Qubits). Mapping classical data points into n-qubit Quantum States.



Data Loading

Data Loading is a function which maps classical data point into n-qubit quantum states.

$$f: 2^n \longrightarrow n$$

Quantum computers naturally map data into Hilbert space. The map that performs the embedding has been termed a quantum feature map.

Quantum Feature Map

Encoding data into quantum states is a feature map.

Feature map transform the data into a new space where it is linear. New space is called Hilbert Space.

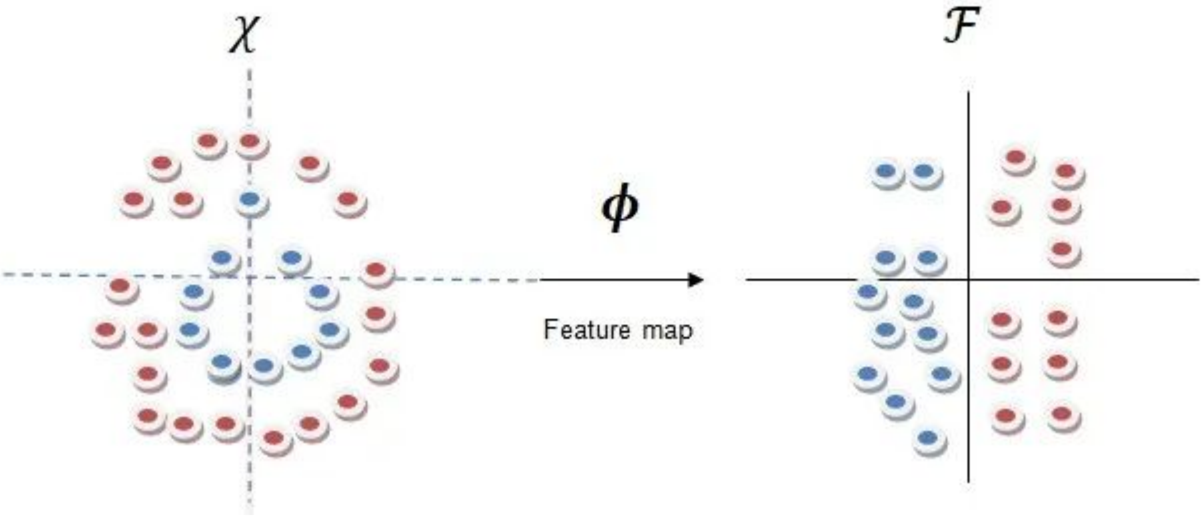
A function which maps input data into feature space.

$$\phi : \chi \longrightarrow \mathcal{F}$$

Where ϕ is a feature map, χ set of Input data, $\mathcal{F}$ Feature space i.e., Hilbert space

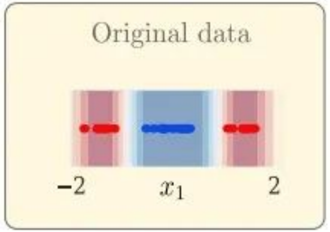
Quantum Feature Map

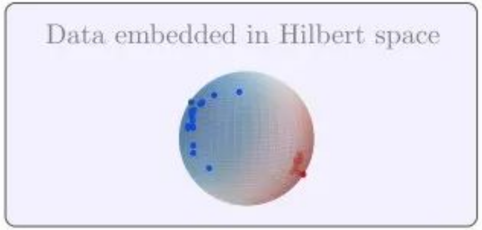
The outputs of the map on the individual data points are called feature vectors.



Quantum Embeddings

A Quantum embedding is then the representation of classical data points ‘x’ as quantum states via a quantum feature map.

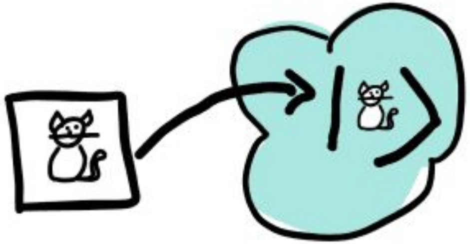




It takes a classical datapoint and translates it into a set of gate parameters in a quantum circuit, creating a quantum state.

Quantum Embeddings

A Quantum embedding is then the representation of classical data points ‘x’ as quantum states via a quantum feature map.



It takes a classical datapoint and translates it into a set of gate parameters in a quantum circuit, creating a quantum state.

Basis Encoding

Basis embedding associates each input with a computational basis state of a qubit system.

The embedded quantum state is the bit-wise translation of a binary string.

x=1001 is represented by the 4-qubit quantum state $|1001\rangle$

Basis Encoding

Let's consider classical input data consisting of M examples, with N features each,

$$\mathcal{D} = x^{(1)}, \ldots, x^{(m)}, \ldots, x^{(M)},$$

where $x^{(m)}$ is a N-dimensional vector for $m = 1, \ldots, M$.

Basis Embedding

Let's consider the classical dataset $\mathcal{D}$ mentioned above. For basis embedding, each example has to be a N-bit binary string; $x^{(m)} = (b_1, \ldots, b_N)$ with $b_i \in 0, 1$ for $i = 1, \ldots, N$. Assuming all features are represented with unit binary precision (one bit), each input example $x^{(m)}$ can be directly mapped to the quantum state $|x^{(m)}\rangle$. **This means that the number of quantum subsystems, n , must be at least equal to $\mathbf{N}$**. An entire dataset can be represented in superpositions of computational basis states as

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^{(m)}\rangle.$$

For example, let's say we have a classical dataset containing two examples $x^{(1)} = 01$ and $x^{(2)} = 11$. The corresponding basis encoding uses two qubits to represent $|x^{(1)}\rangle = |01\rangle$ and $|x^{(2)}\rangle = |11\rangle$, resulting in

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

Basis Encoding

Note

For N bits, there are 2^N possible basis states. Given $M \ll 2^N$, the basis embedding of $\mathcal{D}$ will be sparse.

Amplitude Embedding

In the amplitude-embedding technique, data is encoded into the amplitudes of a quantum state.

A normalized classical N-dimensional data point x is represented by the amplitudes of a n-qubit quantum state as

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle$$

Amplitude Embedding

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle,$$

where $N = 2^n$, x_i is the i-th element of x, and $|i\rangle$ is the i-th computational basis state. In this case, however, x_i can have different numeric data types, e.g., integer or floating point. For example, let's say we want to encode the four-dimensional floating-point array $x = (1.0, 0.0, -5.5, 0.0)$ using amplitude embedding. The first step is to normalize it, i.e., $x_{norm} = \frac{1}{\sqrt{31.25}}(1.0, 0.0, -5.5, 0.0)$. The corresponding amplitude encoding uses two qubits to represent x_{norm} as

$$|\psi_{x_{norm}}\rangle = \frac{1}{\sqrt{31.25}}\left[|00\rangle - 5.5|10\rangle\right].$$

Amplitude Embedding

Let's consider the classical dataset $\mathcal{D}$ mentioned above. Its amplitude embedding can be easily understood if we concatenate all the input examples $x^{(m)}$ together into one vector, i.e.,

$$\alpha = C_{norm} x_1^{(1)}, \ldots, x_N^{(1)}, x_1^{(2)}, \ldots, x_N^{(2)}, \ldots, x_1^{(M)}, \ldots, x_N^{(M)}$$

where C_{norm} is the normalization constant; this vector must be normalized $|\alpha|^2 = 1$. The input dataset can now be represented in the computational basis as

$$|\mathcal{D}\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle,$$

where α_i are the elements of the amplitude vector α and $|i\rangle$ are the computational basis states. The number of amplitudes to be encoded is $N \times M$. As a system of n qubits provides 2^n amplitudes, **amplitude embedding requires $\mathbf{n} \geq \log_2(\mathbf{NM})$ qubits.**

Amplitude Embedding

Note

If the total number of amplitudes to embed, i.e., $N \times M$, is less than 2^n, *non-informative* constants can be *padded* to α (Schuld & Petruccione (2018))

For example, if we have 3 examples with 2 features each, we have $3 \times 2 = 6$ amplitudes to embed. However, we have to use at least $\lceil \log_2(6) \rceil = 3$ qubits, with $2^3 = 8$ available states. We therefore have to concatenate $2^3 - 6 = 2$ constants at the end of α.

Angle Embedding

The simplest type of encoding for floating-point data is AngleEmbedding.

This type of embedding encodes a single floating-point value X ∈ R into a quantum state with the mapping

$$x \mapsto R_k(x)|0\rangle$$

where

K ∈ x, y , z is the axis of rotation in the Bloch sphere.

Default axis of rotation = X

You may also choose to set it to k=y

Avoid k=z

Angle Embedding

Keep in mind that Pauli rotations are 2π-periodic up to a global phase, meaning that you should normalize your data to be in $\Omega := [0, \pi] \subset \mathbb{R}$.

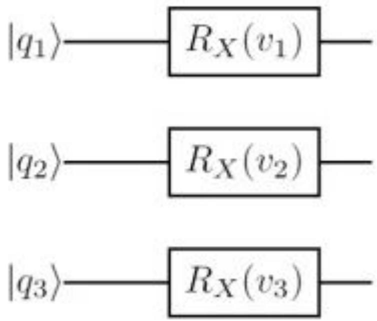
This can be helpful in order to avoid encoding two different values as the same quantum state.

While the AngleEmbedding allows you to encode a lot of information in a single qubit, this comes at the cost of a difficult construction process.

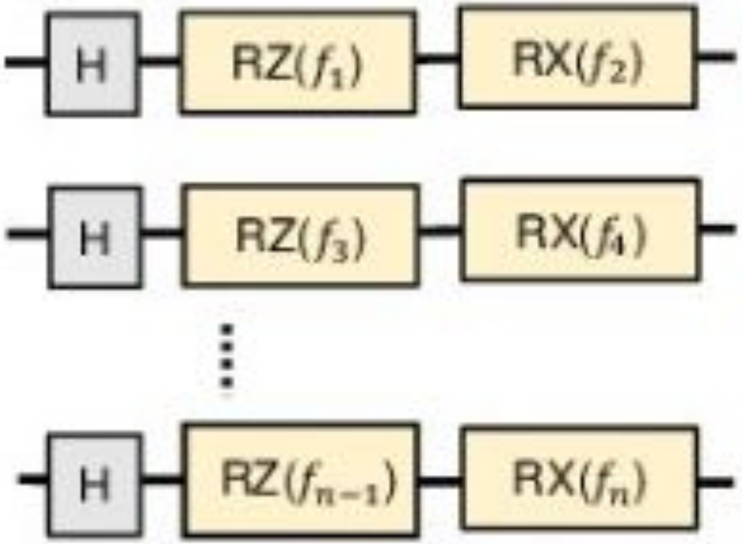
Angle Embedding

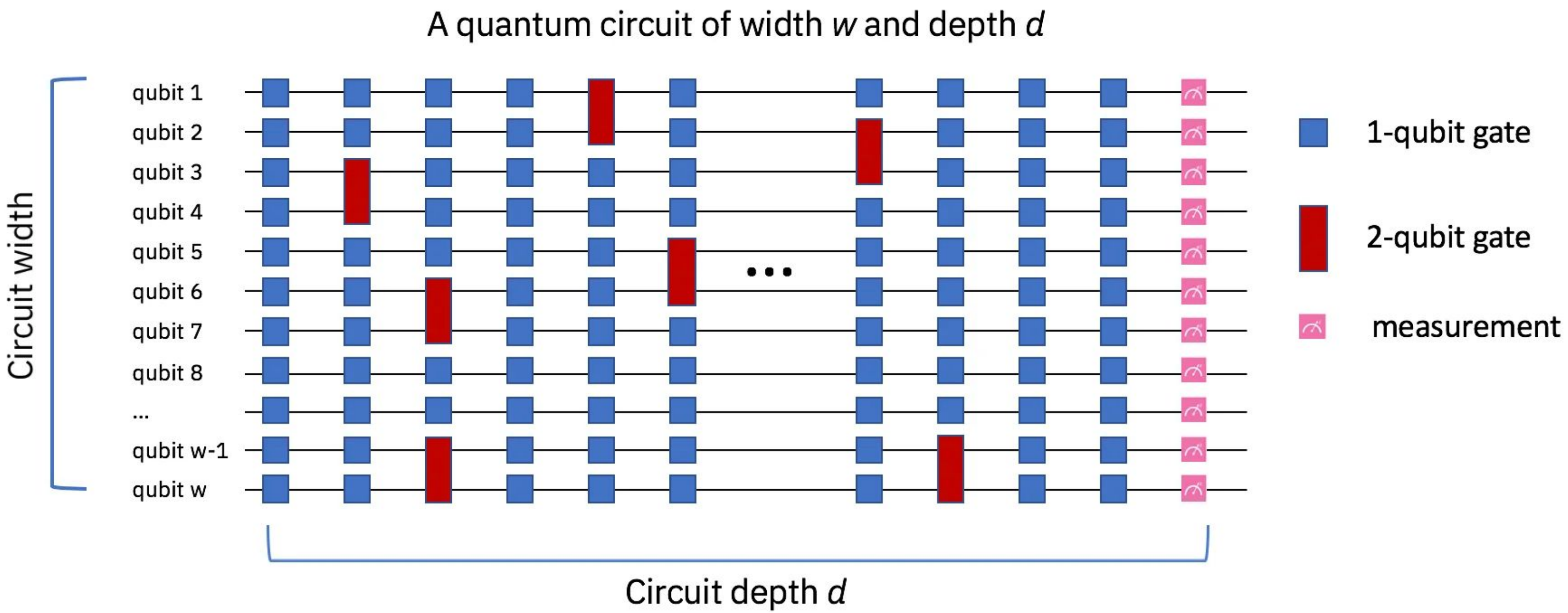
Angle embedding is not robust.
If we wanna apply angle embedding on a dataset the number of rotations will be the same as the number of features in the dataset.

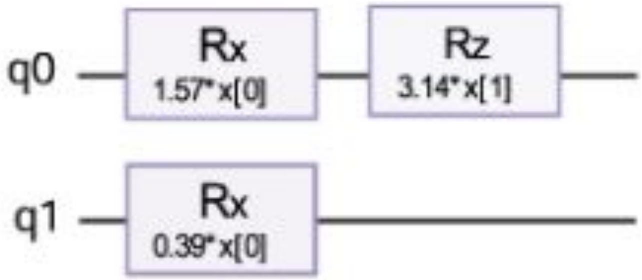
n-dimensional sample would take n-number of qubits to generate the set of quantum states.

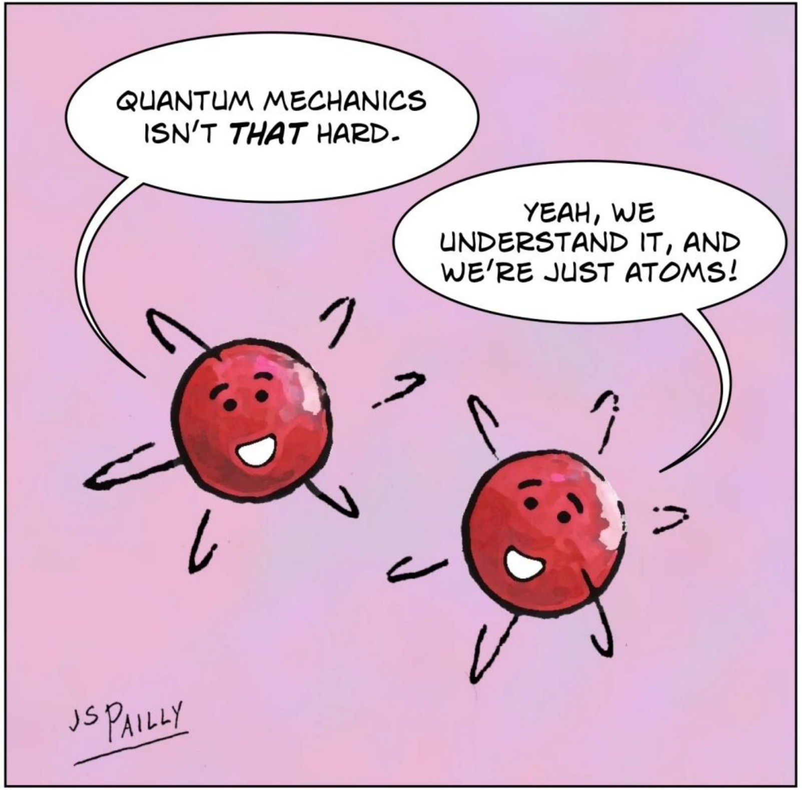


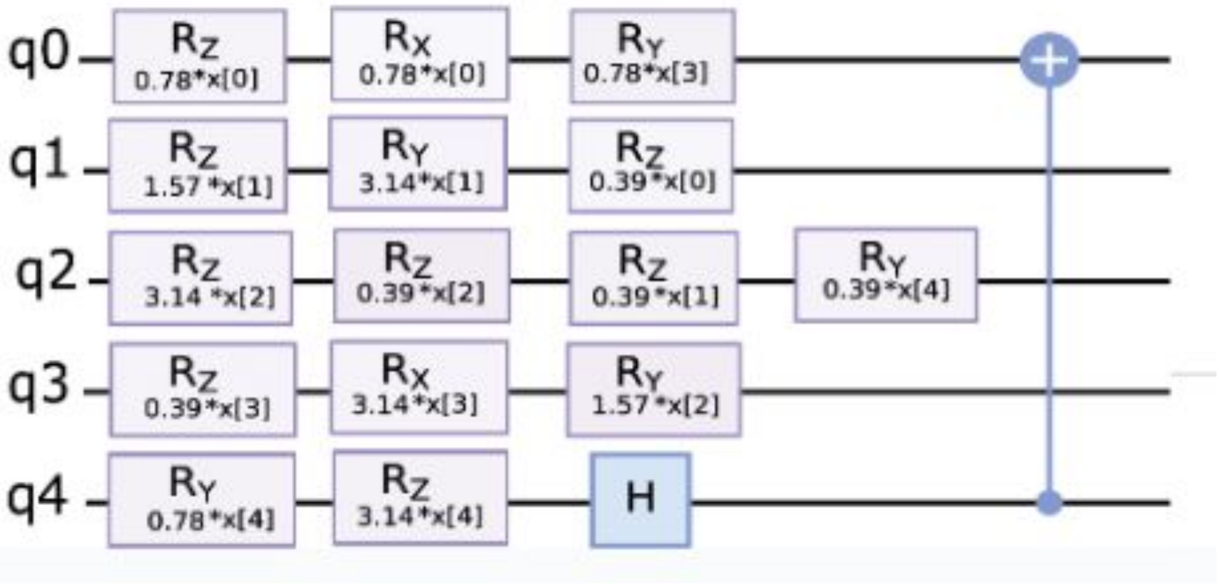
Repeated Angle Embedding

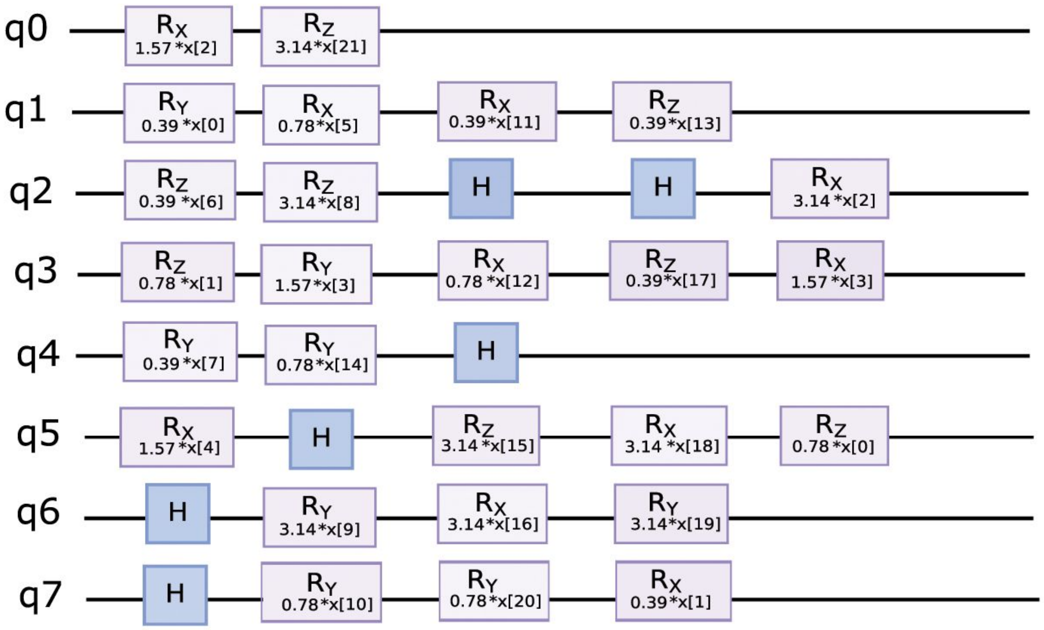



A quantum circuit of width w and depth d
Circuit width
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
qubit 6
qubit 7
qubit 8
...
qubit w-1
qubit w
1-qubit gate
2-qubit gate
measurement
Circuit depth d


q0
Rx
1.57*x[0]
Rz
3.14*x[1]
q1
Rx
0.39*x[0]


QUANTUM MECHANICS ISN'T *THAT* HARD.
YEAH, WE UNDERSTAND IT, AND WE'RE JUST ATOMS!
JS PAILLY


q0
q1
q2
q3
q4
RZ 0.78*x[0]
RX 0.78*x[0]
RY 0.78*x[3]
RZ 1.57*x[1]
RY 3.14*x[1]
RZ 0.39*x[0]
RZ 3.14*x[2]
RZ 0.39*x[2]
RZ 0.39*x[1]
RY 0.39*x[4]
RZ 0.39*x[3]
RX 3.14*x[3]
RY 1.57*x[2]
RY 0.78*x[4]
RZ 3.14*x[4]
H



q0
R_X 1.57*x[2]
R_Z 3.14*x[21]
q1
R_Y 0.39*x[0]
R_X 0.78*x[5]
R_X 0.39*x[11]
R_Z 0.39*x[13]
q2
R_Z 0.39*x[6]
R_Z 3.14*x[8]
H
H
R_X 3.14*x[2]
q3
R_Z 0.78*x[1]
R_Y 1.57*x[3]
R_X 0.78*x[12]
R_Z 0.39*x[17]
R_X 1.57*x[3]
q4
R_Y 0.39*x[7]
R_Y 0.78*x[14]
H
q5
R_X 1.57*x[4]
H
R_Z 3.14*x[15]
R_X 3.14*x[18]
R_Z 0.78*x[0]
q6
H
R_Y 3.14*x[9]
R_X 3.14*x[16]
R_Y 3.14*x[19]
q7
H
R_Y 0.78*x[10]
R_Y 0.78*x[20]
R_X 0.39*x[1]

Pauli Feature map

The Pauli Expansion circuit is a data encoding circuit that transforms input data $\vec{x} \in \mathbb{R}^n$, where n is the `feature_dimension` , as

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \in \mathcal{I}} \phi_S(\vec{x}) \prod_{i \in S} P_i\right).$$

Here, S is a set of qubit indices that describes the connections in the feature map, $\mathcal{I}$ is a set containing all these ndex sets, and $P_i \in \{I, X, Y, Z\}$. Per default the data-mapping ϕ_S is

$$\phi_S(\vec{x}) = \begin{cases} x_i \text{ if } S = \{i\} \\ \prod_{j \in S}(\pi - x_j) \text{ if } |S| > 1 \end{cases}.$$

Pauli Feature Map

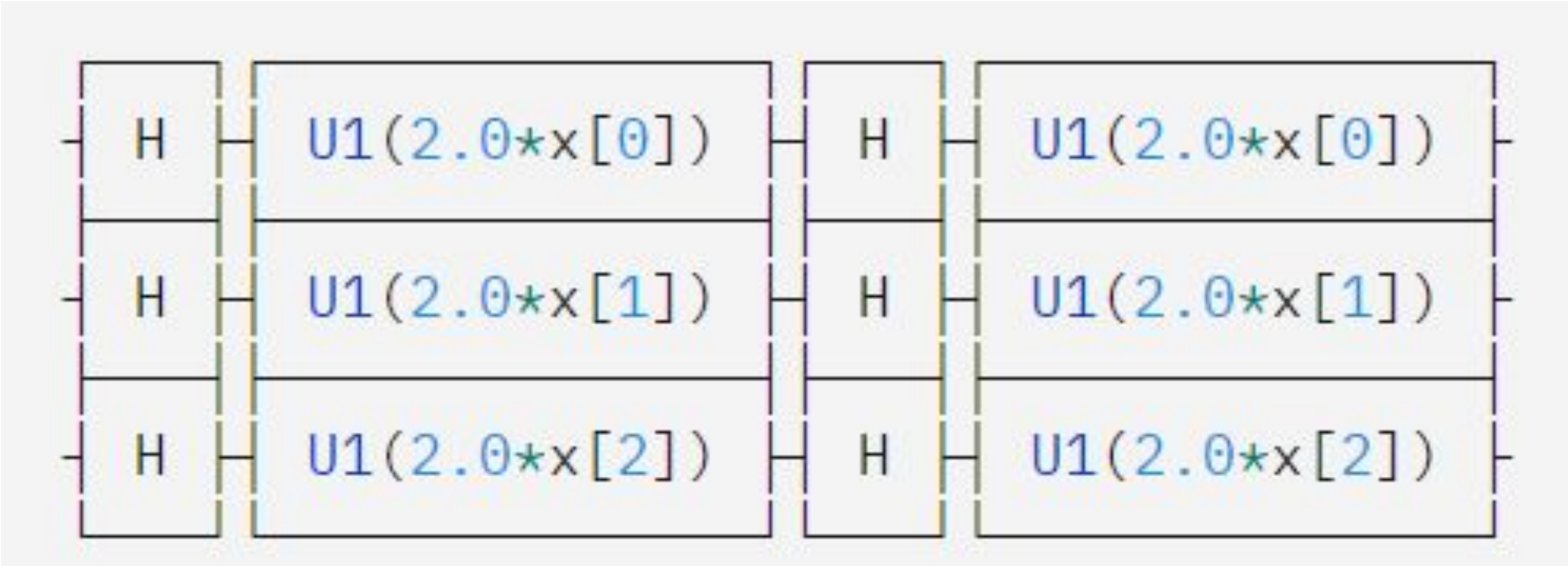
```
>>> prep = PauliFeatureMap(2, reps=1, paulis=['ZZ'])
>>> print(prep)
     ┌───┐
q_0: ┤ H ├──■──────────────────────────────────────■──
     ├───┤┌─┴─┐┌────────────────────────────────┐┌─┴─┐
q_1: ┤ H ├┤ X ├┤ U1(2.0*(pi - x[0])*(pi - x[1])) ├┤ X ├
     └───┘└───┘└────────────────────────────────┘└───┘
```

Pauli Feature Map

```
>>> prep = PauliFeatureMap(2, reps=1, paulis=['Z', 'XX'])
>>> print(prep)
     ┌───┐┌──────────────┐┌───┐                                               ┌───┐
q_0: ┤ H ├┤ U1(2.0*x[0]) ├┤ H ├──■─────────────────────────────────────────■──┤ H ├
     ├───┤├──────────────┤├───┤┌─┴─┐┌─────────────────────────────────┐┌─┴─┐├───┤
q_1: ┤ H ├┤ U1(2.0*x[1]) ├┤ H ├┤ X ├┤ U1(2.0*(pi - x[0])*(pi - x[1])) ├┤ X ├┤ H ├
     └───┘└──────────────┘└───┘└───┘└─────────────────────────────────┘└───┘└───┘
```

Z Feature Map

$$U1(\lambda) = \begin{pmatrix} e^{-i\frac{\lambda}{2}} & 0 \\ 0 & e^{i\frac{\lambda}{2}} \end{pmatrix}$$



ZZ Feature Map

$$U1(\lambda) = \begin{pmatrix} e^{-i\frac{\lambda}{2}} & 0 \\ 0 & e^{i\frac{\lambda}{2}} \end{pmatrix}$$

```
>>> from qiskit.circuit.library import ZZFeatureMap
>>> prep = ZZFeatureMap(2, reps=1)
>>> print(prep)
     ┌───┐┌──────────────┐
q_0: ┤ H ├┤ U1(2.0*x[0]) ├──■─────────────────────────────────────────■──
     ├───┤├──────────────┤┌─┴─┐┌─────────────────────────────────┐┌─┴─┐
q_1: ┤ H ├┤ U1(2.0*x[1]) ├┤ X ├┤ U1(2.0*(pi - x[0])*(pi - x[1])) ├┤ X ├
     └───┘└──────────────┘└───┘└─────────────────────────────────┘└───┘
```

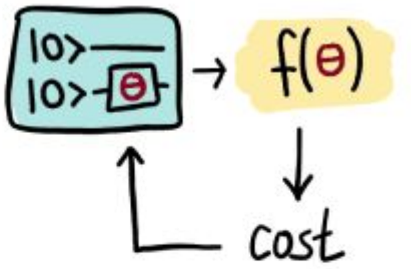
Variational circuits

Variational circuits are also known as "parametrized quantum circuits".

Adaptable quantum circuits

Variational circuits are quantum algorithms that depend on free parameters.

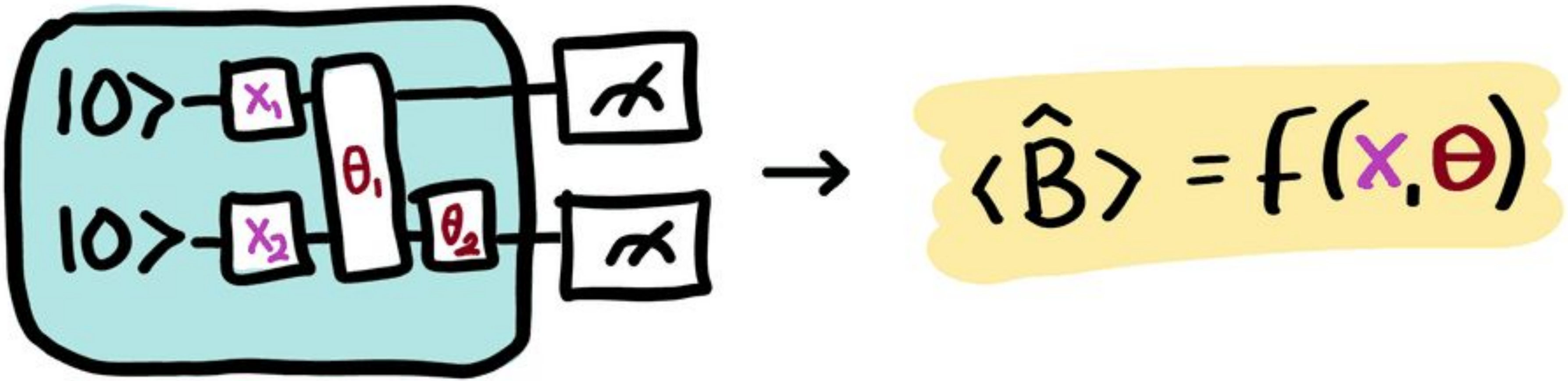
Variational circuits



They consist of three ingredients:

- Preparation of a fixed initial state (e.g., the vacuum state or the zero state)
- A quantum circuit $U(\theta)$, parameterized by a set of free parameters θ.
- Measurement of an observable B at the output. This observable may be made up from local observables for each wire in the circuit, or just a subset of wires.

Building The Circuit



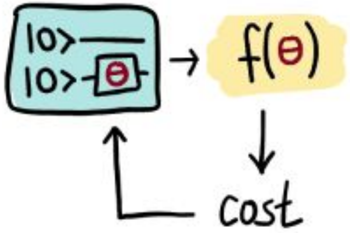
Variational circuits

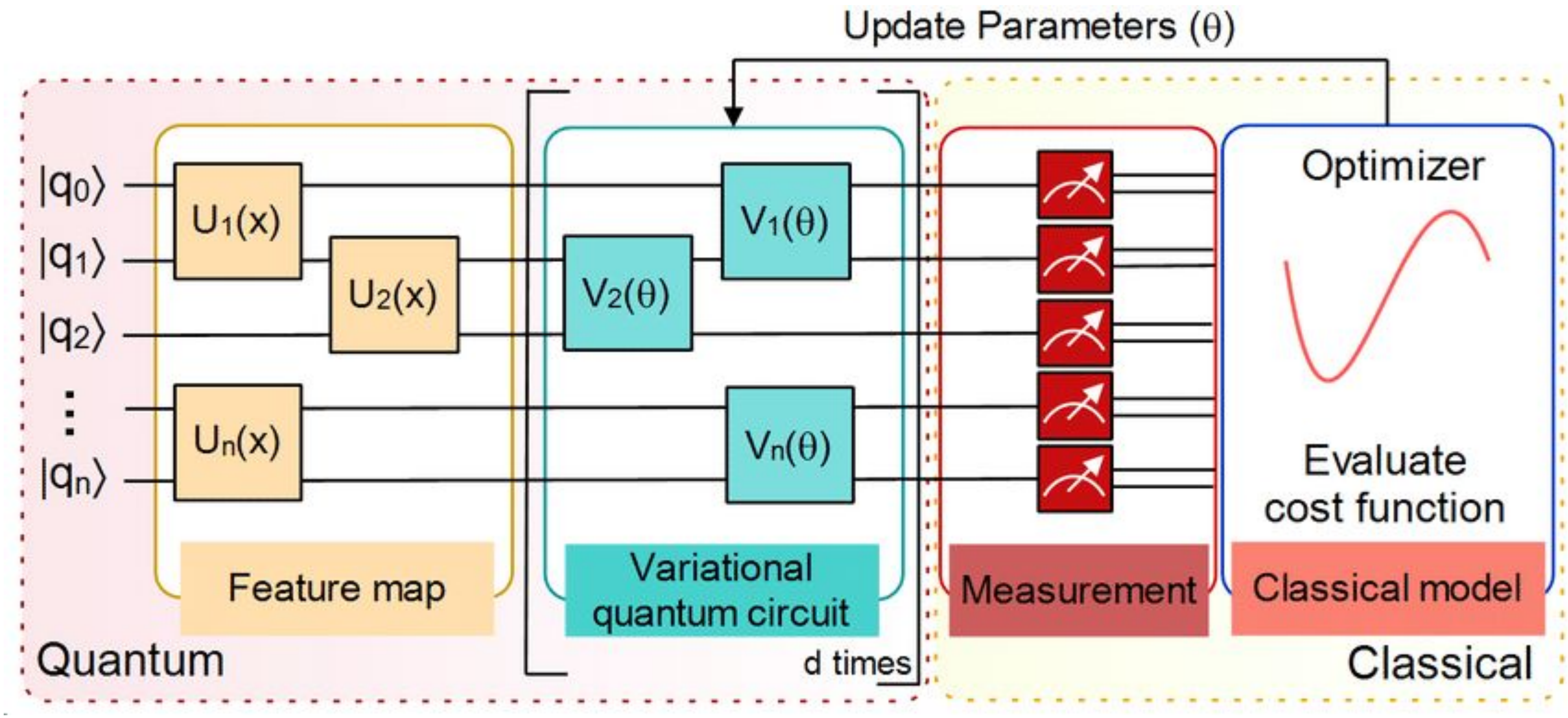
Consider a variational quantum classifier which uses two variational circuits:

The first circuit associates the gate parameters with fixed data inputs

The second circuit depends on free, trainable parameters

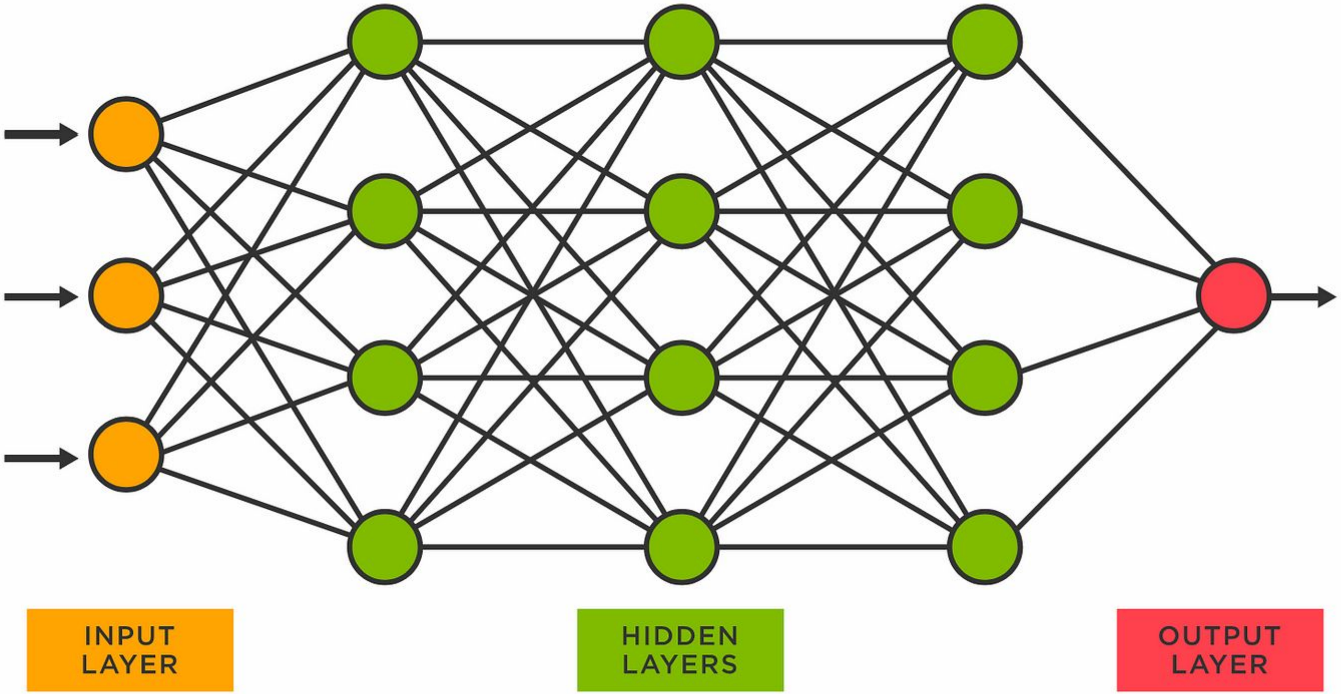
Together with a final measurement, this setup can be interpreted as a machine learning model



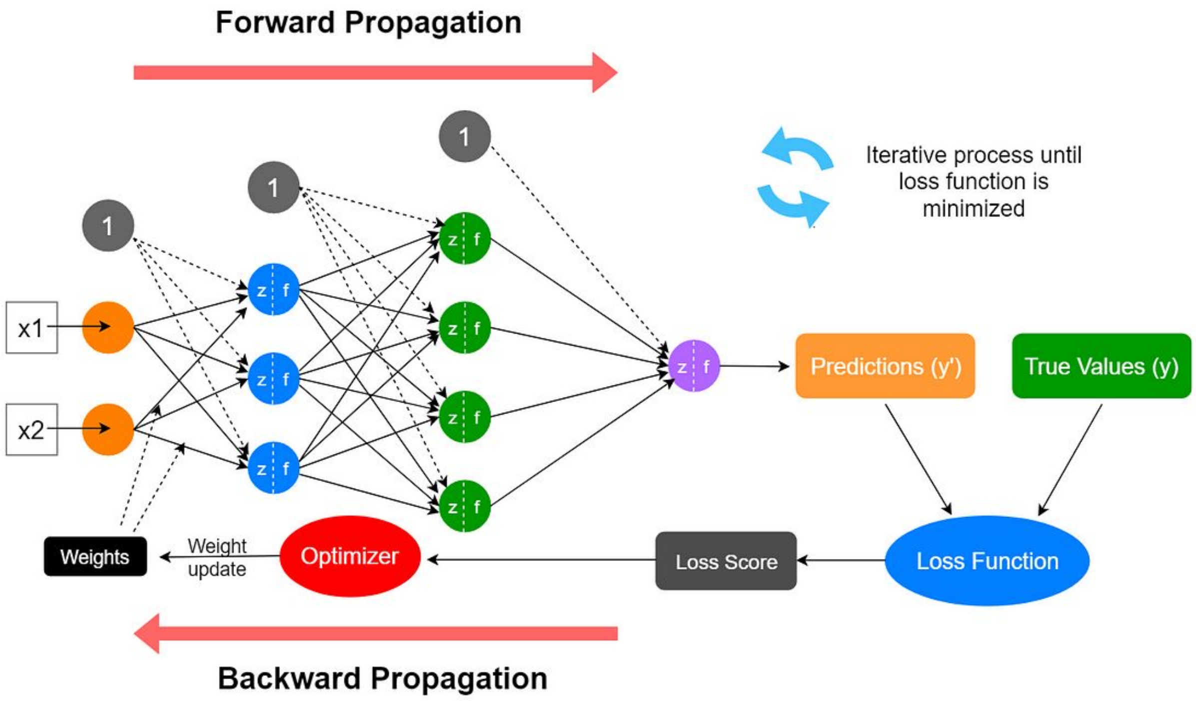


Update Parameters (θ)
|q0⟩
|q1⟩
|q2⟩
|qn⟩
U1(x)
U2(x)
Un(x)
V2(θ)
V1(θ)
Vn(θ)
Optimizer
Evaluate
cost function
Feature map
Variational
quantum circuit
Measurement
Classical model
d times
Quantum
Classical

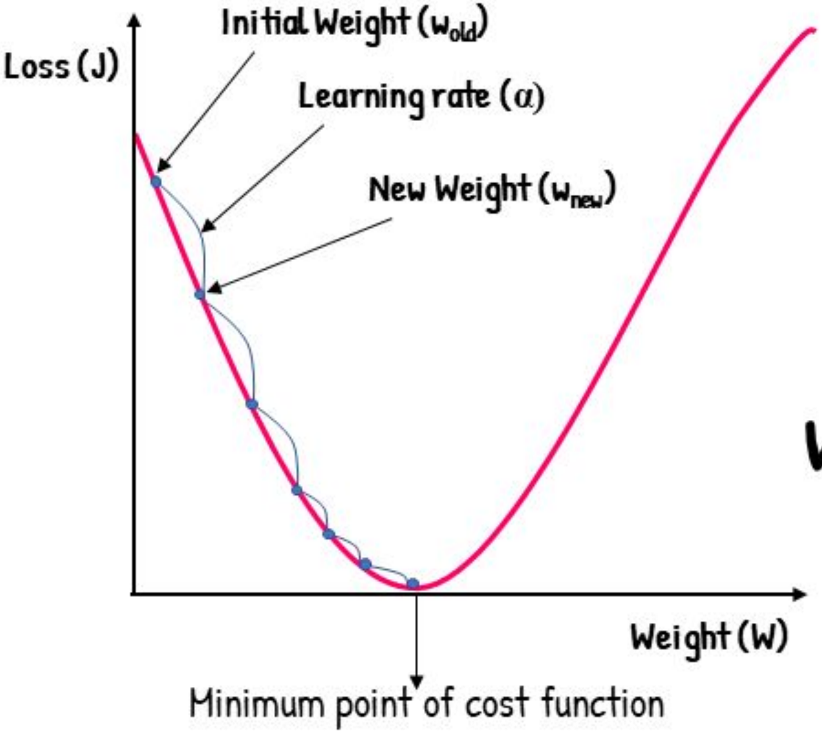
Classical Neural Network



Classical Neural Network



Gradient Descent



$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

Layer 1

$$
\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_{11}^{(1)}} &= \overbrace{\overbrace{\overbrace{\left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)}^{\delta_1^{(3)}}\overbrace{\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)}^{\theta^{(2)}}}^{\text{“proportional error”}}\overbrace{\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)}^{f'\left(a^{(2)}\right)}}^{\delta^{(2)}}\overbrace{\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)}^{\text{input}} + \overbrace{\overbrace{\overbrace{\left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)}^{\delta_2^{(3)}}\overbrace{\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)}^{\theta^{(2)}}}^{\text{“proportional error”}}\overbrace{\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)}^{f'\left(a^{(2)}\right)}}^{\delta^{(2)}}\overbrace{\left(\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}}\right)}^{\text{input}} \\
\frac{\partial J(\theta)}{\partial \theta_{12}^{(1)}} &= \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_1^{(2)}}\right)\left(\frac{\partial \mathrm{a}_1^{(2)}}{\partial z_1^{(2)}}\right)\left(\frac{\partial z_1^{(2)}}{\partial \theta_{12}^{(1)}}\right) \\
\frac{\partial J(\theta)}{\partial \theta_{21}^{(1)}} &= \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}}\right) + \left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{21}^{(1)}}\right) \\
\frac{\partial J(\theta)}{\partial \theta_{22}^{(1)}} &= \underbrace{\left(\frac{\partial J(\theta)}{\partial \mathrm{a}_1^{(3)}}\right)\left(\frac{\partial \mathrm{a}_1^{(3)}}{\partial z_1^{(3)}}\right)\left(\frac{\partial z_1^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}}\right)}_{\text{derivative chain for } \delta_1^{(3)}} + \underbrace{\left(\frac{\partial J(\theta)}{\partial \mathrm{a}_2^{(3)}}\right)\left(\frac{\partial \mathrm{a}_2^{(3)}}{\partial z_2^{(3)}}\right)\left(\frac{\partial z_2^{(3)}}{\partial \mathrm{a}_2^{(2)}}\right)\left(\frac{\partial \mathrm{a}_2^{(2)}}{\partial z_2^{(2)}}\right)\left(\frac{\partial z_2^{(2)}}{\partial \theta_{22}^{(1)}}\right)}_{\text{derivative chain for } \delta_2^{(3)}}
\end{aligned}
$$

Variational circuits

Variational Circuits are the practical embodiment of the idea:

“Let’s train our quantum computers like we train our neural networks”

Also known as Parameterized Quantum Circuits (PQC) & Quantum Neural Networks (QNN)

Variational Quantum Algorithm

Variational Quantum Algorithm

A VQA is an algorithmic framework that uses variational methods, typically employing one or more VQCs, to minimize (or maximize) a cost function.

A VQA optimizes the parameters of the VQC to achieve specific goals, such as minimizing the energy of a quantum state or solving a combinatorial optimization problem.

Variational Quantum Algorithm

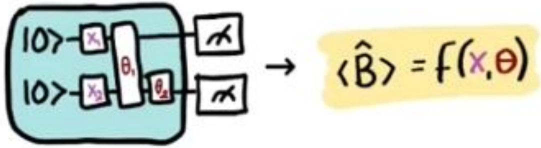
How can today's NISQ (Noisy Intermediate Scale Quantum) devices be optimally utilized to achieve quantum advantage?

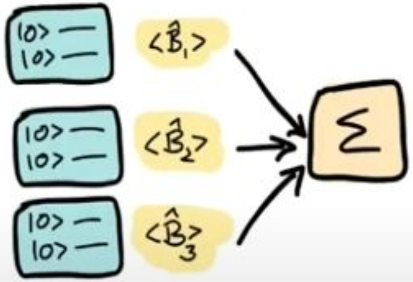
These algorithms adopt the concept of training quantum computers in a similar manner to training neural networks, that is, finding the optimum parameters of the model to minimize/maximize some objective function related to that model.

Variational Quantum Algorithms

A variational algorithm contains a few ingredients:

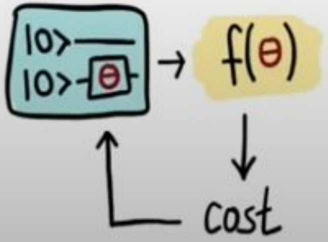
1. A circuit **ansatz**





2. A problem-specific **cost-function**

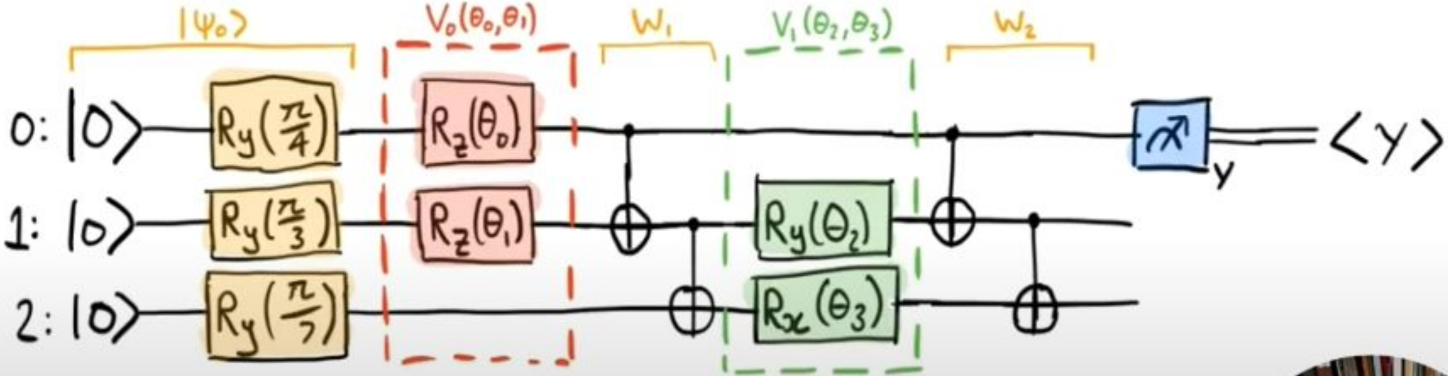
3. A training procedure



Circuit Ansätze

Ansatz:

"an educated guess or an additional assumption made to help solve a problem, and which is later verified to be part of the solution by its results"



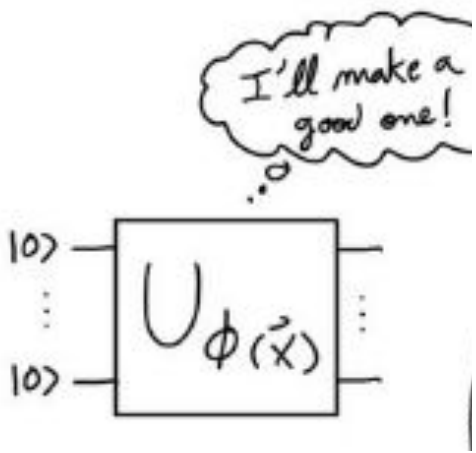
Choosing an Ansatz

The ansatz can come from:

- Some basis in physics, chemistry, or quantum information theory (e.g., VQE)
- The structure of the problem (e.g., QAOA)
- Intuition borrowed from machine learning
- No place at all (use your imagination!)

The choice of ansatz affects the model/function that can be learned (more layers often better)

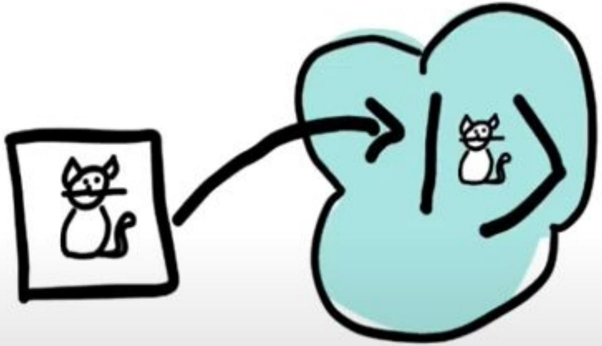
Variational Quantum Algorithm

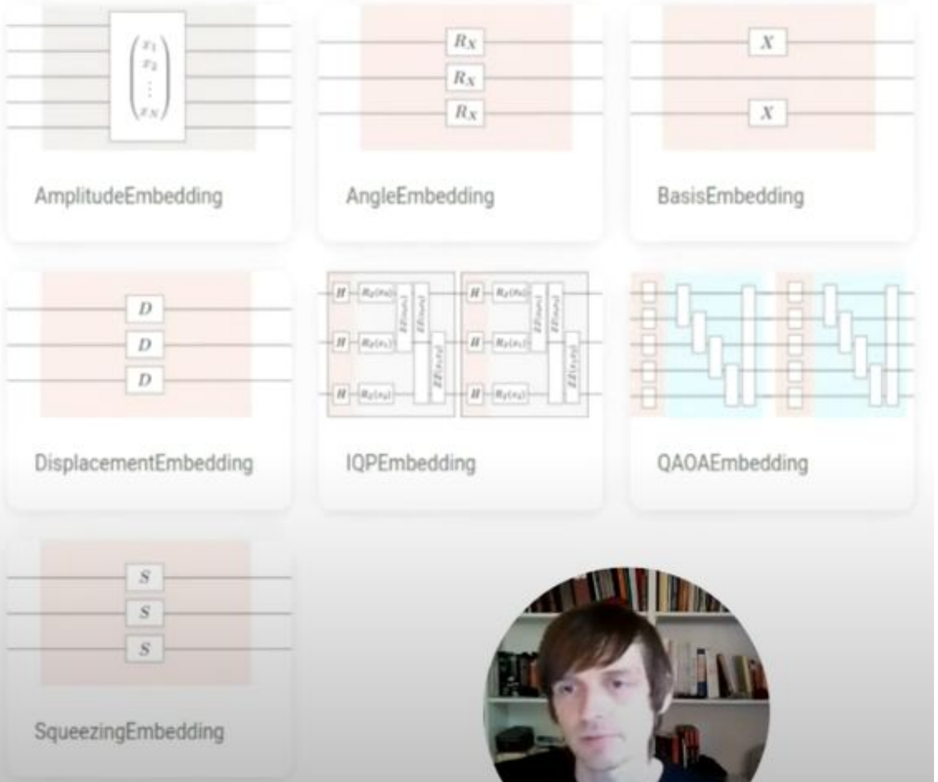


QUEST FOR ANSATZ

Embedding Classical Data

Variational circuits have free parameters, but it is often required to **input or embed** classical data as part of the ansatz





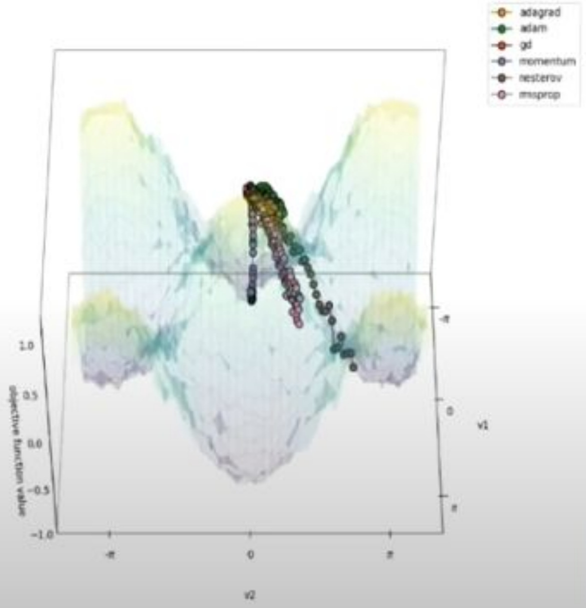
Optimizing Variational Circuits

Using the parameter-shift rule, we can optimize circuits using **gradient descent**

Many flavours: *GD*, *Momentum*, *Nesterov*, *Adagrad*, *RMSProp*, *Adam*, *Newton*, *etc.*

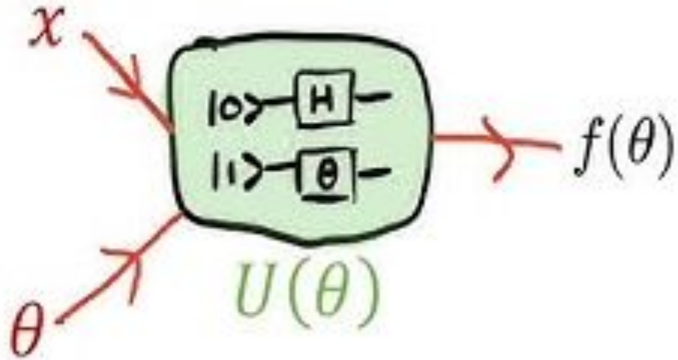
There are also a number of emerging **Quantum-aware optimizers**:

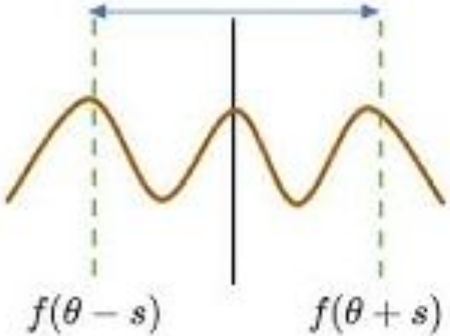
- Rotosolve/Rotoselect
- Quantum Natural Gradient
- iCANS/Rosalin



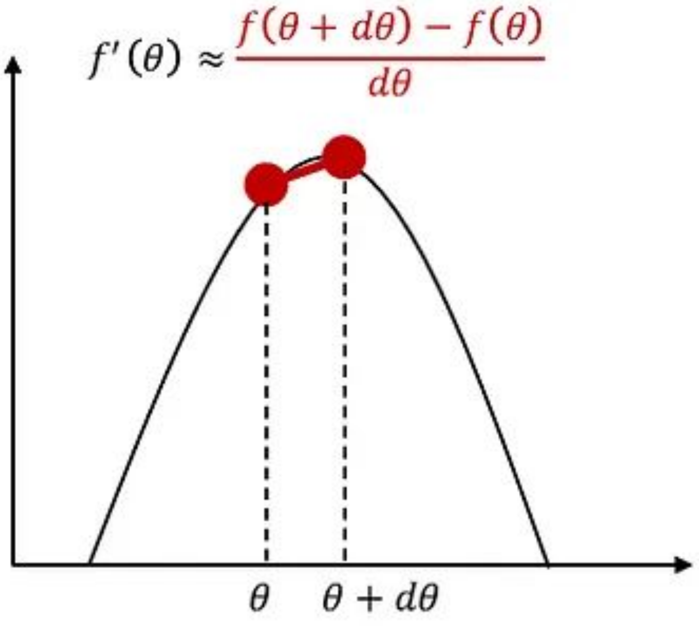
The Parameter-Shift Rule

Fortunately, *quantum circuits admit a parameter-shift rule!*

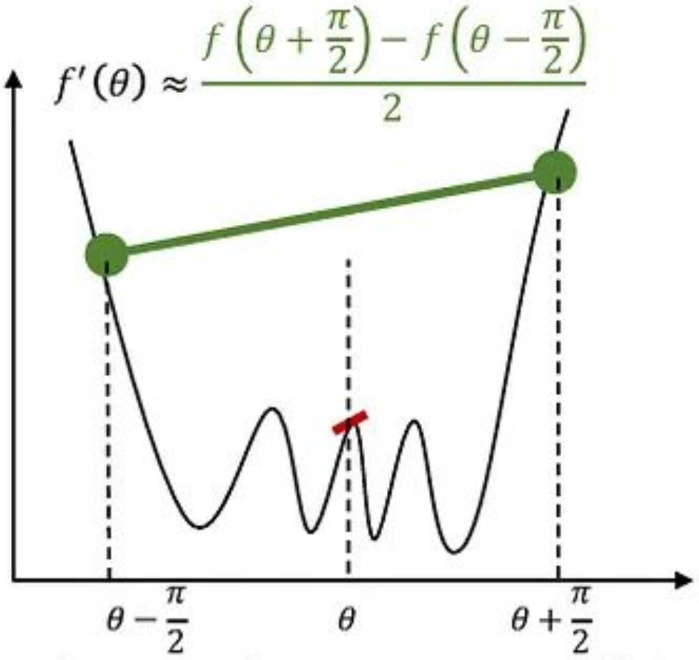




$$\frac{\partial}{\partial\theta} f(\theta) = c[f(\theta + s) - f(\theta - s)]$$



Numerical Gradient



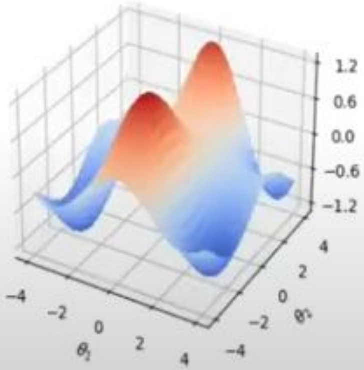
Analytic Gradient (Parameter Shift)

Quantum Aware Optimizers

Rotosolve/Rotoselect

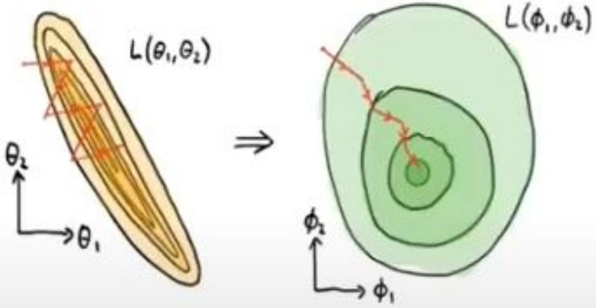
Don't use gradients at all!

Instead, they solve directly for the minimum w.r.t. one coordinate at a time



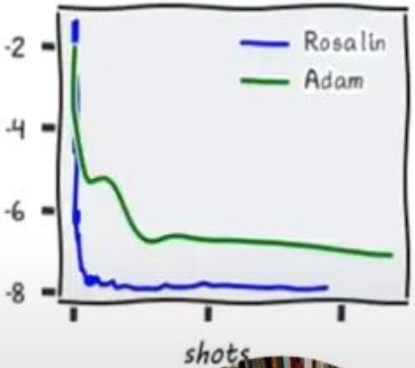
Quantum Natural Gradient

Accounts for the inherent geometry of quantum Hilbert space



iCANS/Rosalin

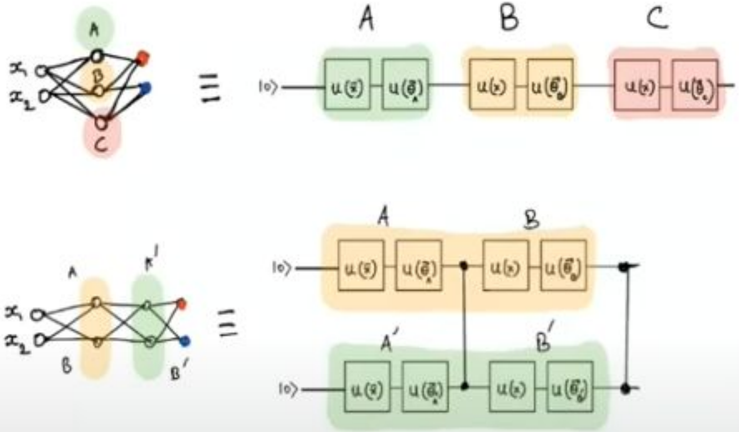
"Shots-frugal" optimizers estimate many quantities using limited samples



Insights on Quantum Embeddings

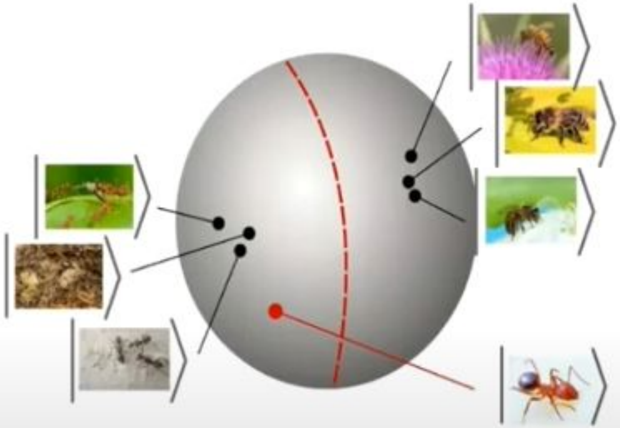
Data "reuploading"

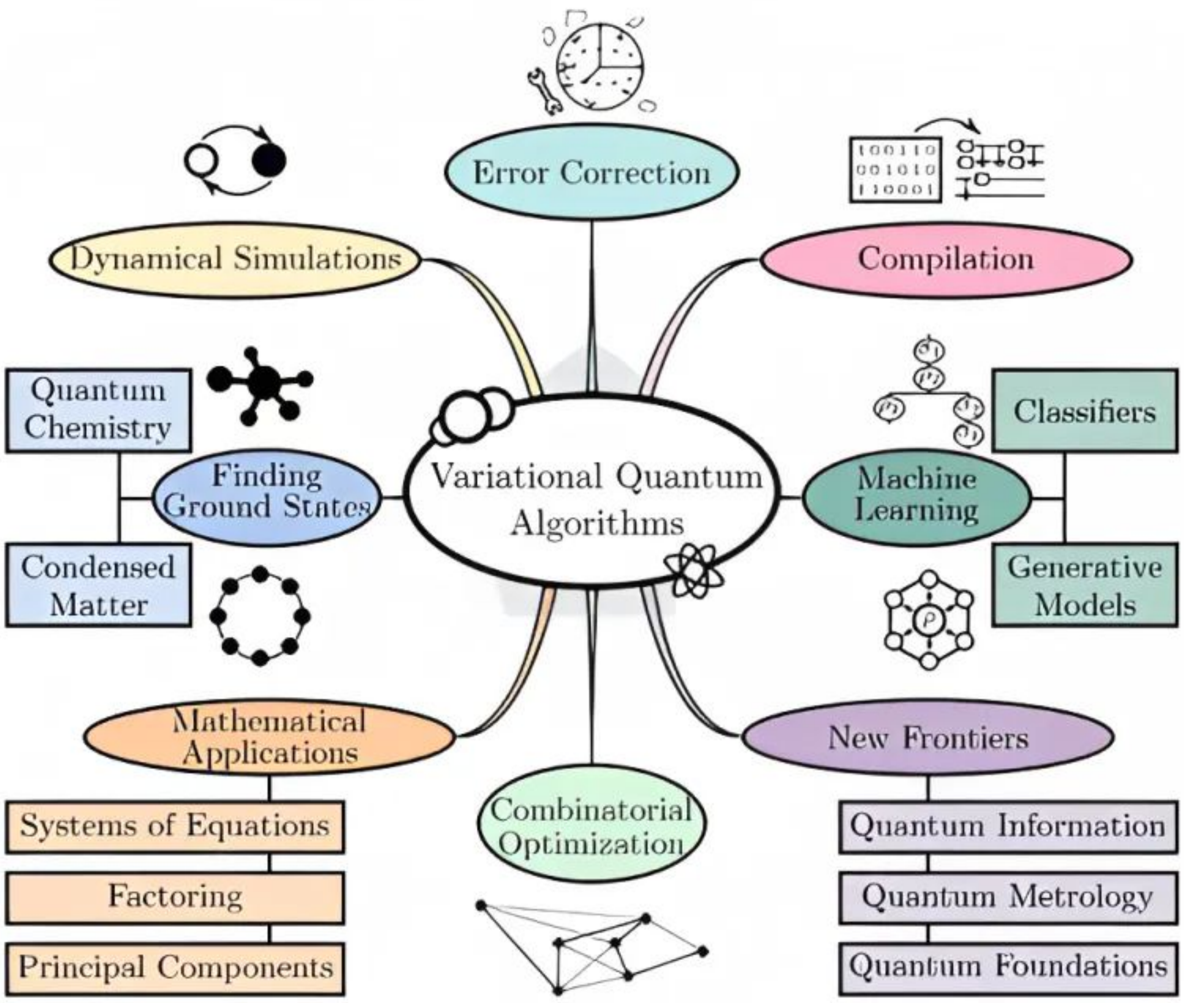
Idea: *repeated sequence of embeddings*



Learned embeddings

Idea: *don't train the circuit, train the embeddings*




Error Correction
Dynamical Simulations
Compilation
Quantum Chemistry
Finding Ground States
Condensed Matter
Variational Quantum Algorithms
Machine Learning
Classifiers
Generative Models
Mathematical Applications
Systems of Equations
Factoring
Principal Components
Combinatorial Optimization
New Frontiers
Quantum Information
Quantum Metrology
Quantum Foundations

References for VQC

https://pennylane.ai/qml/glossary/variational_circuit/#variational-circuits

https://www.youtube.com/watch?v=YtepXvx5zdI

https://medium.com/@darrenmarston34/quantum-gradient-descent-v-s-classical-gradient-descent-1f69f5a22cb7

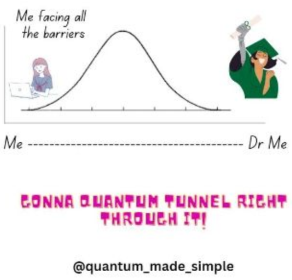
quantum_made_simple Following Message

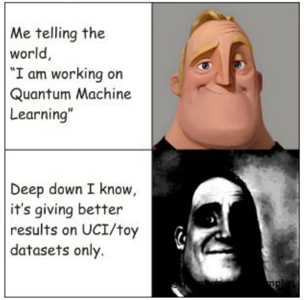
17 posts **67** followers **2** following

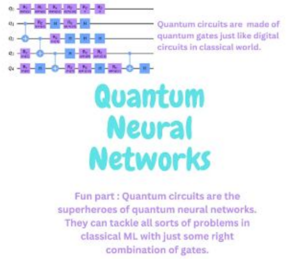
Quantum Made Simple
Quantum for everyone!

Followed by **javeria.ilyas.21**, **_saffia_baloch_** + 40 more

POSTS TAGGED







References

https://shafi-syed.medium.com/quantum-data-and-its-embeddings-1-3b022b2f1245

https://shafi-syed.medium.com/quantum-data-embeddings-circuit-design-2-f2c6f6c89662

https://pennylane.ai/blog/2022/08/how-to-embed-data-into-a-quantum-state/

https://pennylane.ai/qml/glossary/quantum_embedding/

https://www.youtube.com/watch?v=S8zSfxbgEhk