Secrets revealed in this session:

To explore and explain Qiskit facilities to support Quantum Machine Learning in 50 mins

QML and its aims
Parameterised circuits
Variational quantum algorithms
Data encoding and decoding
State measurement
Ansatz design and training
Model geometry and gradients
Parameters optimisation
Curse of dimensionality
QML readings
Qiskit example
Summary and Q&A

See: Ironfrown ABC+BCD Labs (Github)

Exploring
Quantum Machine Learning (with Qiskit)

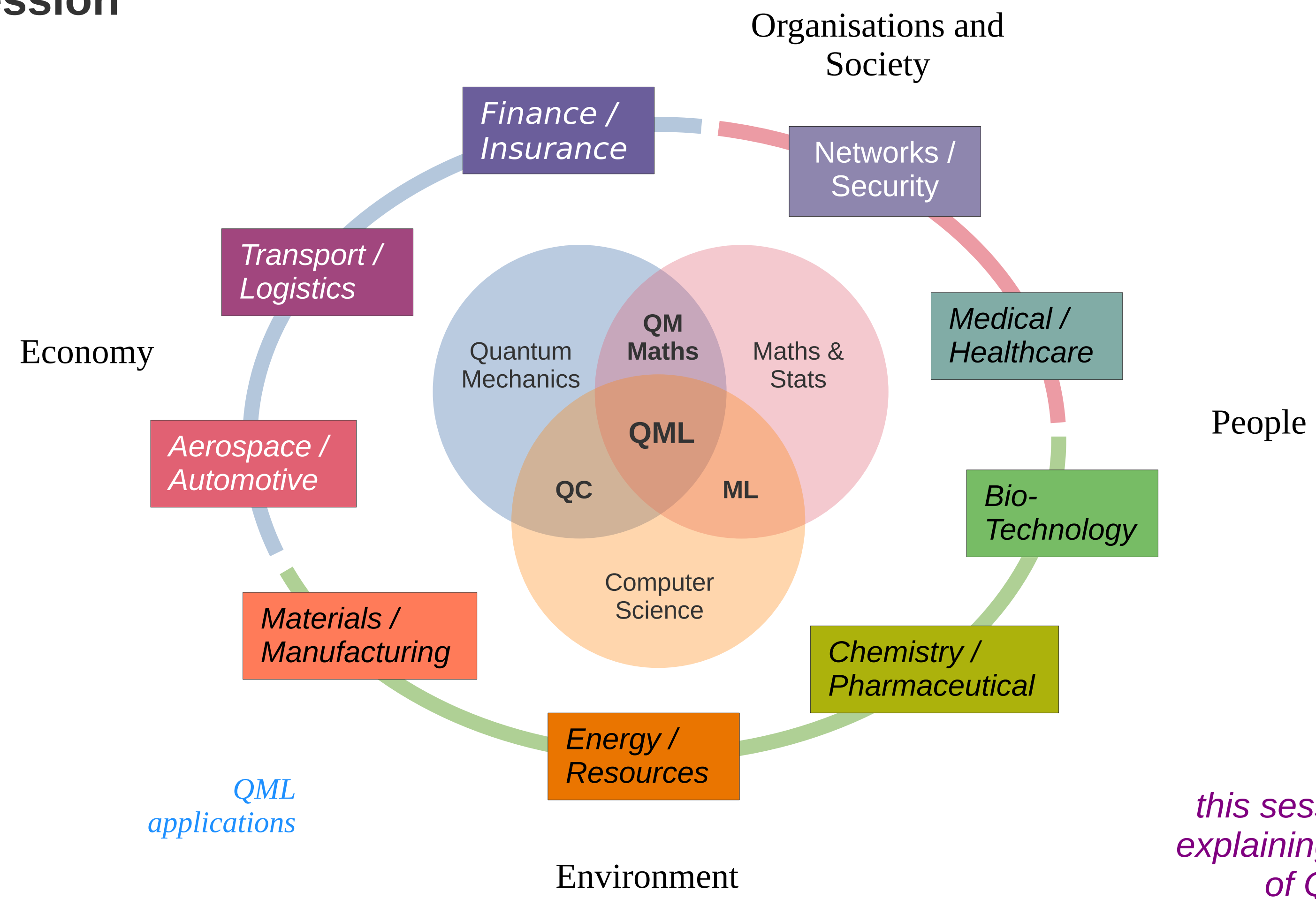
Jacob L. Cybulski
Enquanted, Australia

QISKIT 2025 2025 FALL FEST

Nov 20, 2025, Qiskit Fall Fest, Université Paris-Saclay, Paris, France.

Quantum ML
aims of this session

Jacob Cybulski, Founder
Enquanted, Australia



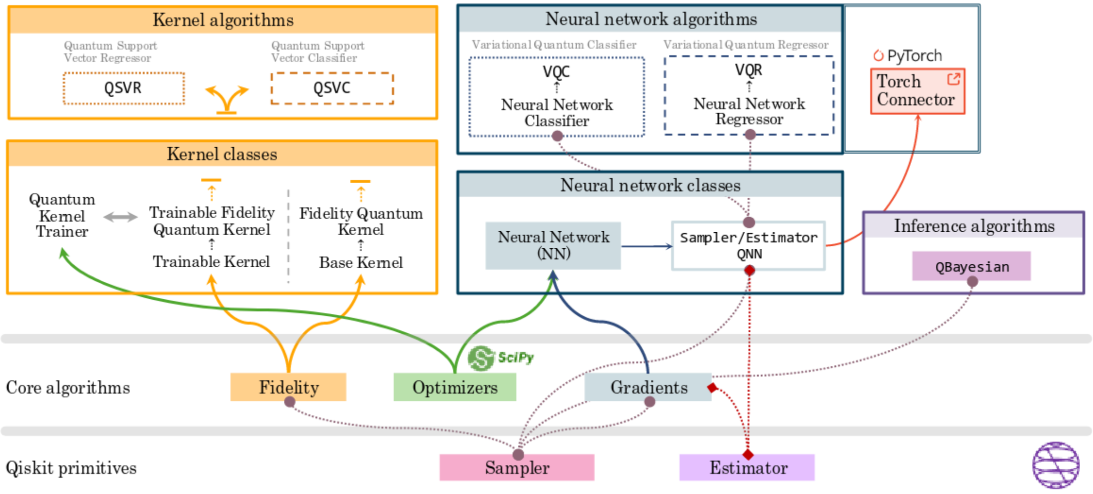
QML applications

this session aims at explaining the nature of QML models

Qiskit and QML

Why Qiskit? *It features…*

- Support for Python, Rust, C++ and more...
- Standard set of quantum state operations
- Execution on simulators and quantum hardware
- Execution on hardware accelerators (e.g. GPUs)
- Tools for error mitigation
- Variety of quantum gradients models
- Support for hybrid quantum-classical computation
- Large community ecosystem (libraries)
- Extensions with PyTorch and TensorFlow
- Hardware agnostic via vendor backends *including* IBM quantum backends and runtime
- Best performer
- High complexity
- Core design changes very often!



Why Qiskit Machine Learning? *Models and tools...*

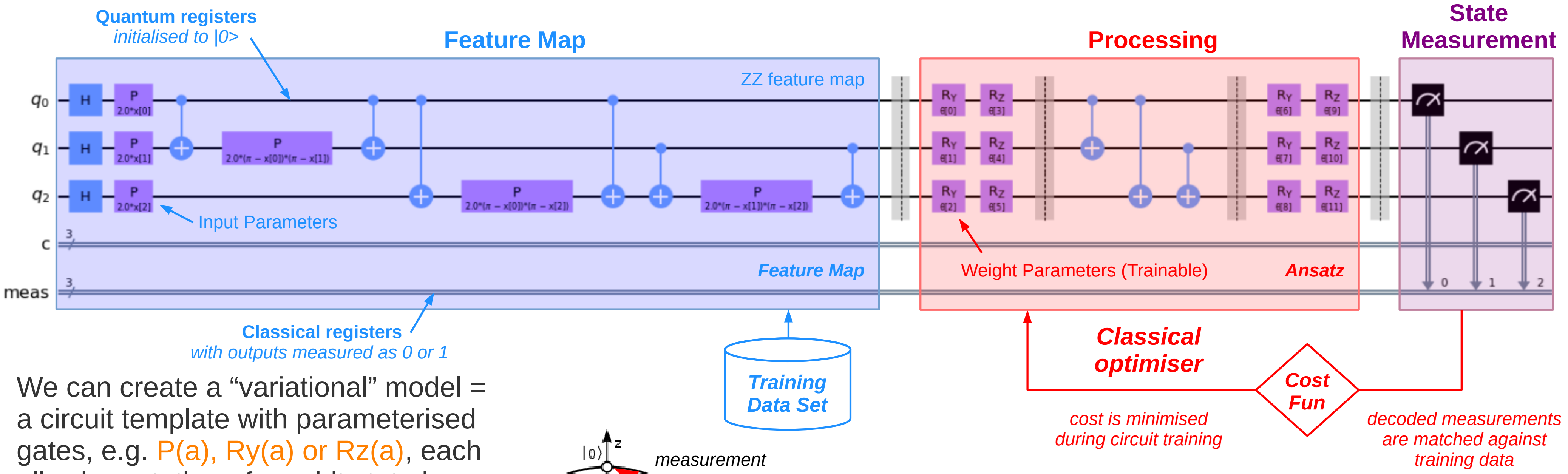
- Quantum Neural Networks (QNN, VQC/R, QCNN, qGAN)
- Quantum Kernel Methods (Feature Maps, Estimators)
- Quantum Support Vector Machines (QSVM, QSVC/R)
- Quantum Bayesian Modelling (Qbayesian)
- Quantum Kernel Principal Components Analysis (QKPCA)
- Quantum Clustering Algorithms (QCA k-NN, DQC)
- Quantum Optimisation Algorithms (QAOA, QUBO)
- Many others available from GitHub and publications...

Sahin, M.E., Altamura, et al., 2025. Qiskit Machine Learning: an open-source library for quantum machine learning tasks at scale on quantum hardware and classical simulators. ArXiv.2505.17756.

Olivier Ezratty, Understanding Quantum Technologies (2025)

Parameterised Quantum Circuits
and Variational Quantum Algorithms

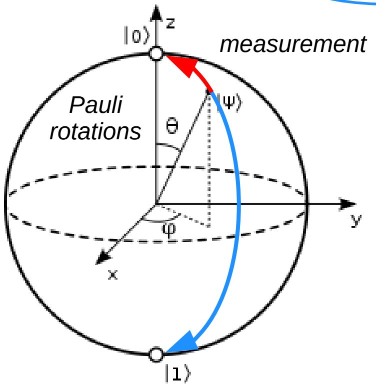
Variational quantum circuits are not executable! They must first be instantiated, i.e. all of their input and weight parameters must be assigned values!



We can create a "variational" model = a circuit template with parameterised gates, e.g. P(a), Ry(a) or Rz(a), each allowing rotation of a qubit state in x, y or z axis (as per Bloch sphere).

Typically (but now always), such circuits consist of three blocks:

- a feature map (input)
- an ansatz (processing)
- measurements (output)



Classical input data is encoded (embedded) into the feature map's parameters, setting the model's initial quantum state.

The quantum state is altered by an ansatz, of parameterised quantum gates, which are trained by a classic optimiser

The circuit final state is measured and decoded (interpreted) as the model's output in the form of classical data.

Data encoding strategies

Feature maps

There are many methods of data embedding, such as: the *basis*, *angle*, *amplitude*, *QRAM*, ... encoding,

In this session we will rely on *angle encoding* realised as qubit state rotation by the angles defined by the data.

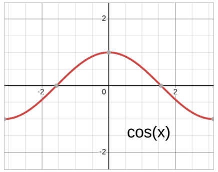
The rotation operators are always available in a quantum platform API, e.g. *Rx*, *Ry*, *Rz*, *P* or *U* (*zyz*).

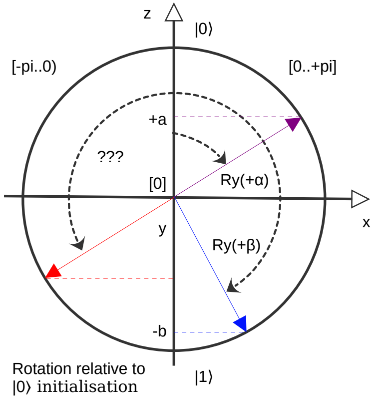
Typically, the encoding rotation is performed around x or y axis, or both (allowing two values per qubit).

Rotations are *relative to a specific qubit state*, commonly starting at |0⟩ state, or (|0⟩+|1⟩)/sqrt(2), which require qubits to be initialised in these states.

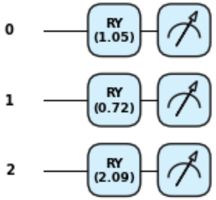
The encoded value could be represented either by the *angular rotation*, or the *amplitude* of the qubit projective measurement (Z).

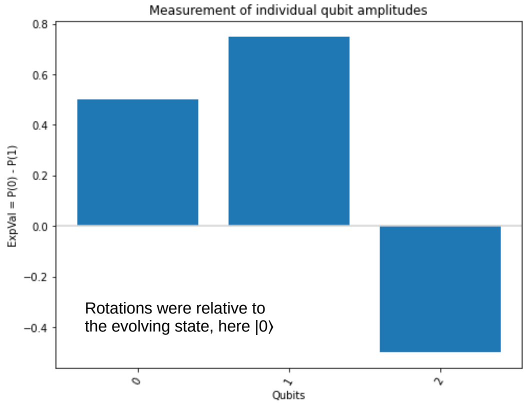
Input data can also be repeatedly encoded and spread around the circuit, which is called *data reuploading*, and which is known to improve the model performance.





Note that training will place qubit states in areas x < 0 and arbitrarily around the z axis. Measurements of such states cannot distinguish them from "pure" x > 0 and z = 0.





Input
Values entered: [np.arccos(0.5), np.arccos(0.75), np.pi-np.arccos(0.5)]
Ry angles used: [1.047, 0.723, 2.094]

Measurements
Probabilities: [[0.25, 0.75], [0.562, 0.438], [0.25, 0.75]]
Amplitudes: [0.5, 0.75, -0.5]

Maria Schuld and Francesco Petruccione.
Machine Learning with Quantum Computers. 2nd ed. Springer, 2021.

Encoding nightmares

The Good, the Bad and the Ugly

Two principles of quantum data encoding:

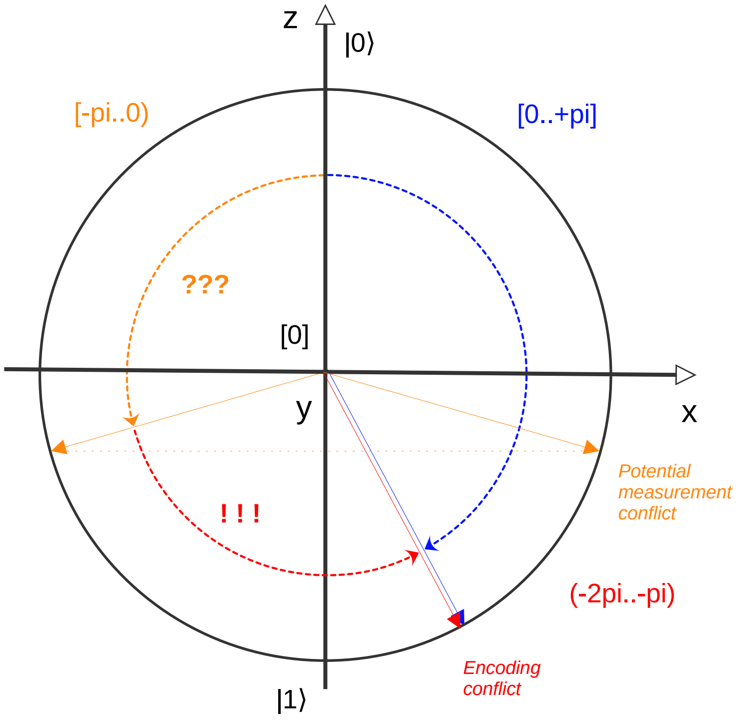
1) **distinct data values should always map onto distinct angles (and possibly also amplitudes)**

2) **the same data values should always map into identical angles (and possibly also amplitudes)**

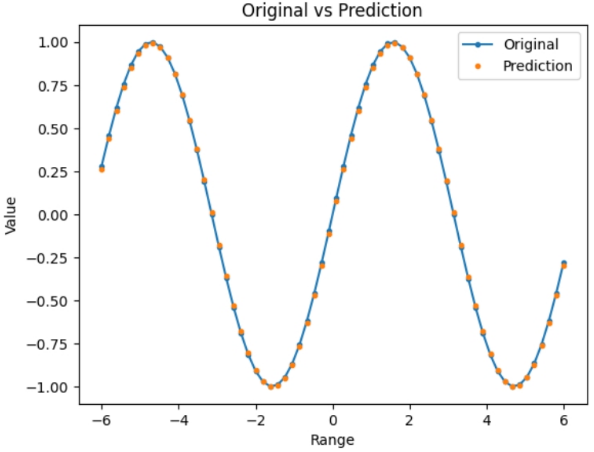
This example shows encoding values wrapping around the Bloch sphere (possibly several times).

This could result in different values to be mapped into the same amplitude (orange), which can be corrected by trainable rotational operations (Ry).

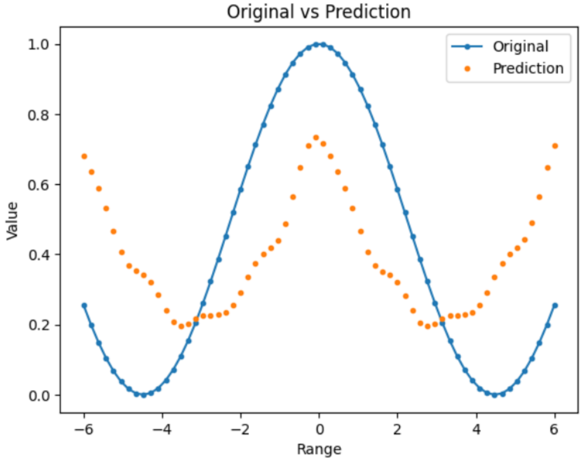
This could also result in different values to be mapped into the same angles (red), which cannot be corrected.



Encoding sine function data in an interval [-2pi, 2pi] will deceptively result in a "good" model (due to data symmetry).



However, even a slight change to the function generating data will make the model impossible to converge.



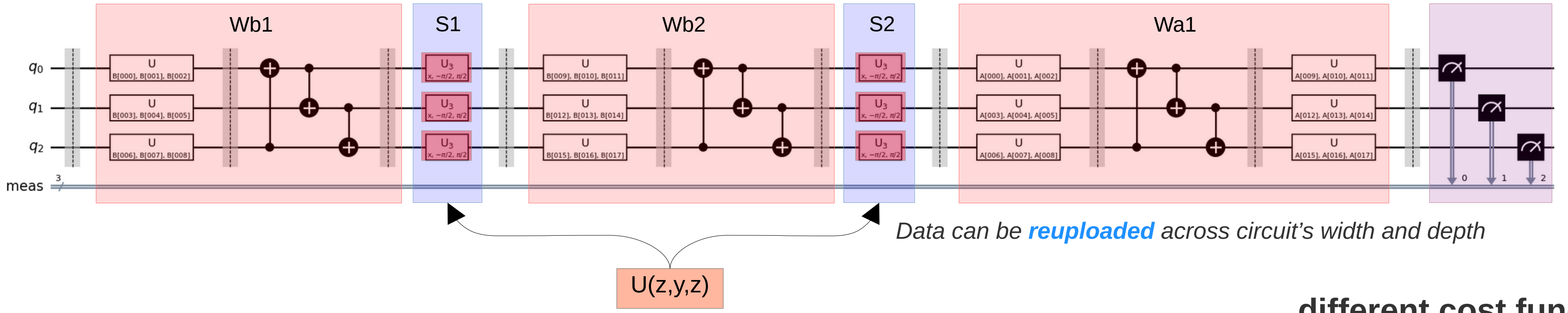
Ansatz design and training

A sample curve fitting model ...

Beware that
adding qubits adds
parameters and entanglements!

The number of states represented by the circuit **grows exponentially** with the number of qubits!

Encoding of classical data in a quantum circuit is
***not** what our ML experience tells us about **inputs** !*



*Data can be **reuploaded** across circuit's width and depth*

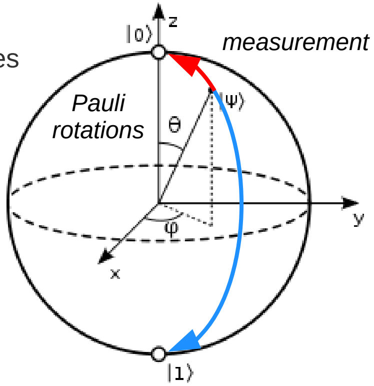
feature maps vary in:
structure and function

ansatze vary in:

- width (qubits #)
- depth (layers #)
- dimensions (param #)
- structure (e.g. funnelling)
- entangling (circular, linear, sca)

ansatz layers consist of:
rotation blocks and entangling blocks of U(z, y, z) and CNOT gates
(rotations) (entanglement)

rotation gates alter qubit states around x, y, z axes



To execute a circuit we just apply it to input data and the optimum parameters

different cost functions:
R2, MAE, MSE, Huber, Poisson, cross-entropy, hinge-embedding, Kullback-Leibner divergence

different optimisers:
gradient based (Adam, NAdam and SPSA)
linear approximation methods (COBYLA)
non-linear approximation methods (BFGS)
quantum natural gradient optimiser (QNG)

circuit execution on:
simulators (CPUs), accelerators (GPUs) and real quantum machines (QPUs)

Commonly used
measurements and their decoding

Quantum circuits can be measured in many ways, e.g.

- all qubits (global cost / measurement)
- a few selected qubits (local cost / measurement)
- groups of qubits (each as a variable value)

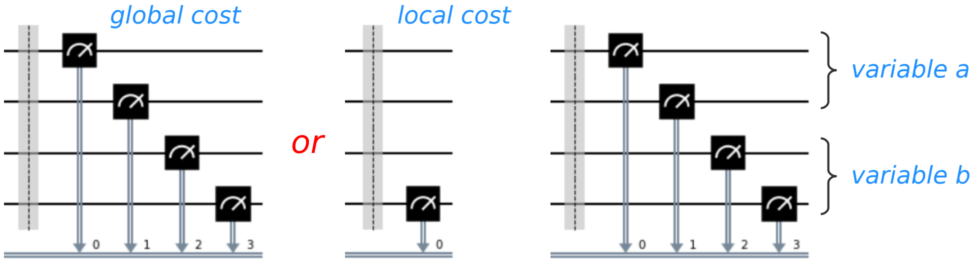
And received in many different formats, e.g.

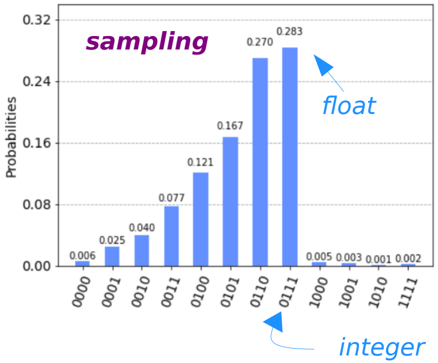
- as counts of outcomes (repeated measurements)
- as probabilities of outcomes (e.g. P(|0111⟩))
- as Pauli expectation values (i.e. of eigenvalues)
- as interpretation of expectation values (e.g. 0..15), etc.

Repeated measurements can be decoded, i.e. interpreted as outcomes of different types, e.g.

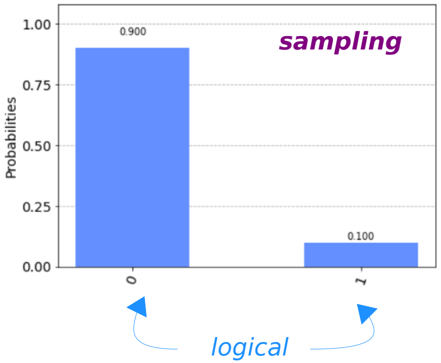
- as a probability distribution (as is)
- as a series of values (via expvals)
- as a binary outcome:
  single qubit measurement or parity of kets
- as an integer:
  most probable ket in multi-qubit measurement
- as a continuous variable:
  expectation value or the probability of a ket (e.g. $|0^n\rangle$)

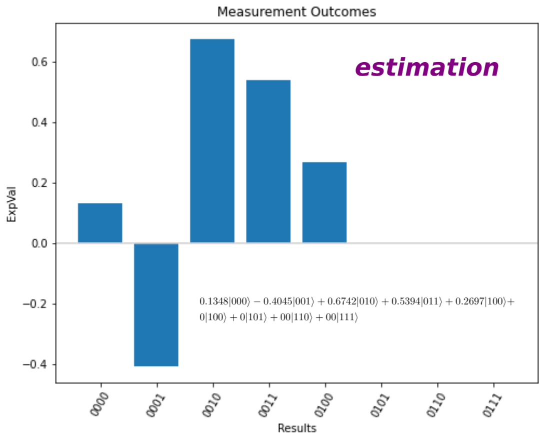
Note that as expected measurement destroys the qubit state, it also loses phase info that is vital for some apps (e.g. QAE)





or





Or we can measure expectation values of the qubits states and interpret them as a series of values in the range [-1..+1]

Beware that **adding 1 measurement → doubles the number of outcomes!**

So... having ***n*** measurements leads to ***2^n*** outcomes

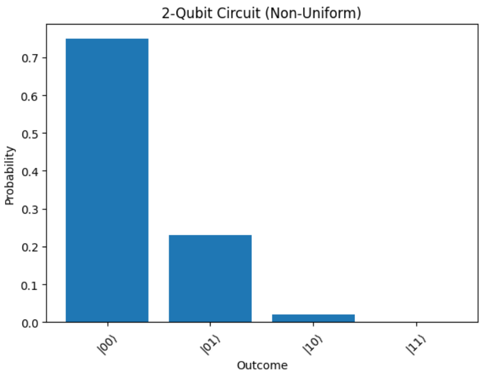
Maria Schuld and Francesco Petruccione. *Machine Learning with Quantum Computers. 2nd ed*. Springer, 2021.

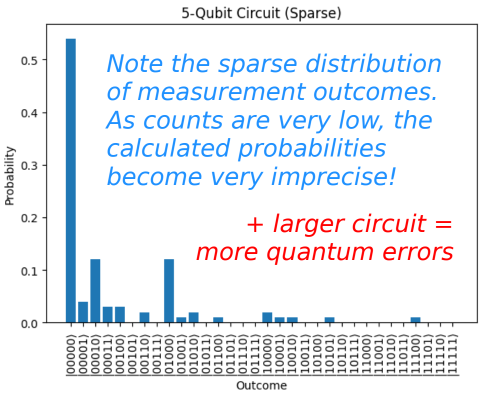
Measurement nightmares

The Good, the Bad and the Ugly

Olivia lanes (2024): “Run Large-Scale Quantum Circuits (100+ Qubits Explained)”, Quantum Computing in Practice, https://www.youtube.com/playlist?list=PLOFEBzvs-VvoZxe2ClFy27yOt6VzsTEJK.

Chris Wood (2024): “An Introduction to Qiskit Runtime Primitives Version 2”, Qiskit Summer School 2024, https://www.youtube.com/watch?v=OuYz02clnx4&t=5s.





Sparse measurement outcomes

(too many measurements, esp. as probability distribution)

- When increasing the number of measurements, we also exponentially increase the number of outcomes, leading to the exponential increase in the number of circuit runs!
- Unless the number of runs is increased with measurements, distribution of outcomes becomes sparse and the probability calculations become imprecise (fig. left).

Working with large circuits (qubits# > 100)

- Instead of working with probability distributions (scales with $O(2^q)$) we need to switch to expectation values (converges with $O(1/\epsilon^2)$)
- When developing an estimator, it is possible to group observables to reduce the number of measurement outcomes
- It is also possible to evaluate a complex Hamiltonian (linear combination of expvals), which can group observables into meaningful outcomes
- Instead of specifying the number of runtime shots (still possible), we instead specify the precision of measurement outcomes

Dealing with errors (large circuits)

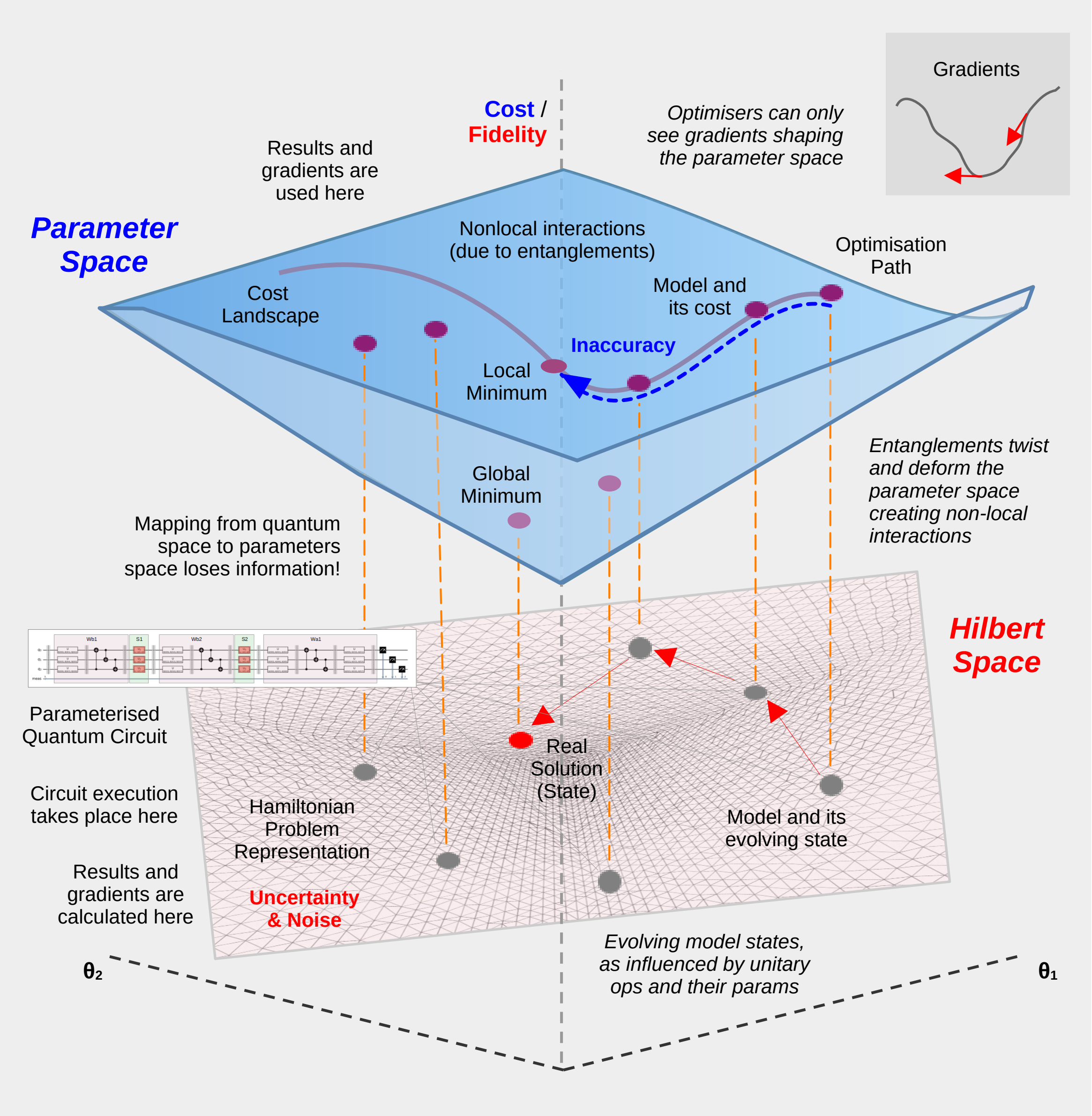
- When working with large circuits, it is also necessary to deploy error suppression and mitigation
- Qiskit allows some of those techniques to be applied automatically at runtime and for specific observables, e.g. *dynamical decoupling*, *twirling* (TREX) and *ZNE / PEC* (Zero Noise Extrapolation / Probabilistic Error Cancellation)

Working with quantum models

Parameter space
(dim = the number of params) is a classical multi-dim space of trainable gate parameters, which the optimiser navigates

Entanglements
(defined by CNOTs) create and correlate non-separable qubit states, which alter the parameter space geometry, and also the cost landscape used by the optimiser

Measurement
of individual qubits collapses their states, consequently projecting the circuit state onto classical outcomes, in the process we lose some quantum info (e.g. phase)



Hilbert state space
(dim ≈ $2^{\text{the number of qubits}}$) is the quantum realm where the models and their states evolve in response to unitary operations as defined by the circuit gates

Data encoding
brings in classical data into the Hilbert space as unique and correlated quantum states during the model execution

Circuit layers
determine the evolution of the quantum model's initial state into its final state during the circuit execution

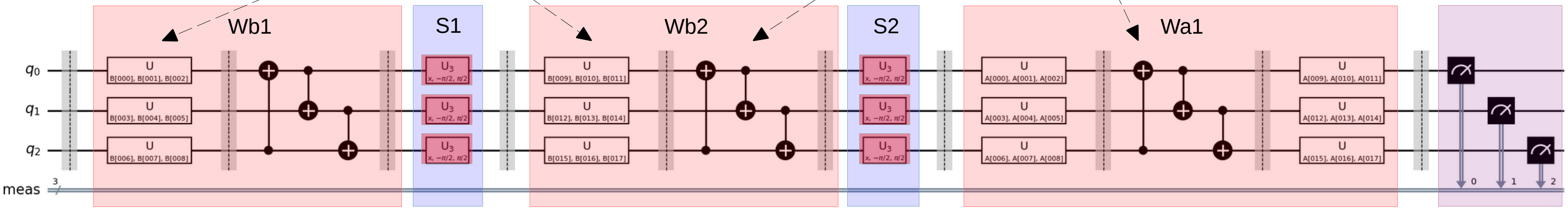
Example model

Expressivity vs Trainability

Parameter space
(dim = the number of params)
is a classical multi-dim space of trainable gate parameters, which the optimiser navigates – that is the only available info to the optimiser!

Data encoding
brings in classical data into the Hilbert space as unique and correlated quantum states during the model execution

Entanglements
(defined by CNOTs) create and correlate non-separable qubit states, which alter the parameter space geometry, and also the cost landscape used by the optimiser, entanglements cause non-local interactions

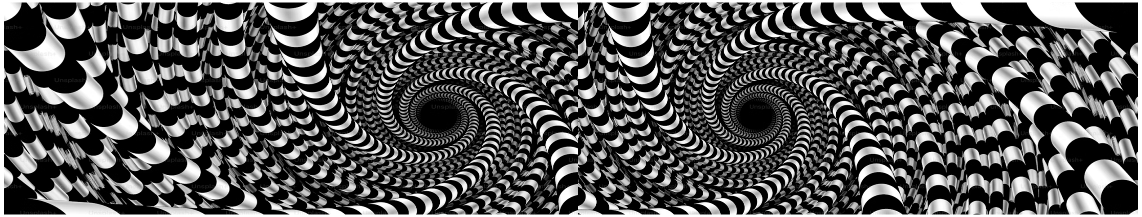


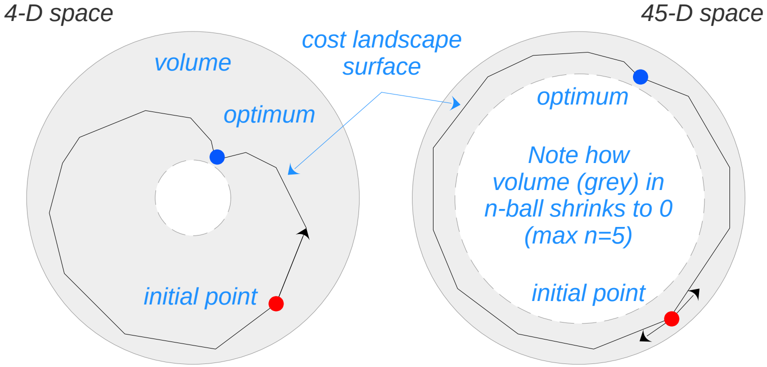
Hilbert state space
(dim ≈ $2^{\text{the number of qubits}}$)
is the quantum realm where the models and their states evolve in response to unitary operations as defined by the circuit gates - this is where the quantum activity takes place!

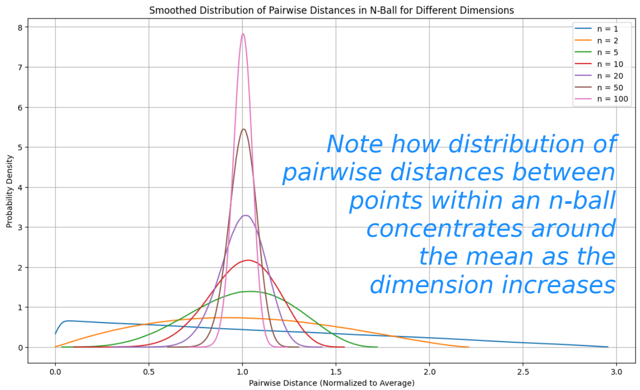
Circuit layers
determine the evolution of the quantum model's initial state into its final state during the circuit execution

Measurement
of individual qubits collapses their states, consequently projecting the circuit state onto classical outcomes, in the process we lose some quantum info (e.g. phase)

The curse
of dimensionality







Cybulski, J.L., Nguyen, T., 2023. “Impact of barren plateaus countermeasures on the quantum neural network capacity to learn”, Quantum Inf Processing 22, 442.

Barren Plateaus (too many dimensions)

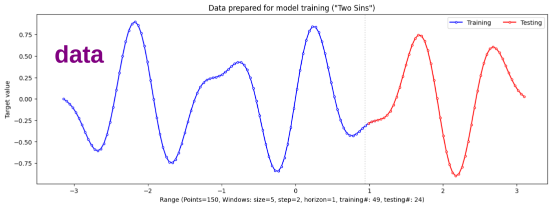
- Pairwise distances between uniformly distributed points in high-dimensional space become (almost) identical, and the surface of such a space is almost flat (n-ball value is near its surface).
- In a quantum model with a high-D parameter space, the cost landscape is nearly flat, the situation called *barren plateau (BP)*.
- In high-D parameter space, models sampled by the optimiser are very sparse in both Hilbert space and parameter space.
- When BPs emerge, the optimiser struggles finding the optimum.
- Selecting the optimisation initial point far from the optimum (e.g. random) makes it even more difficult !

There are some well-known BP countermeasures

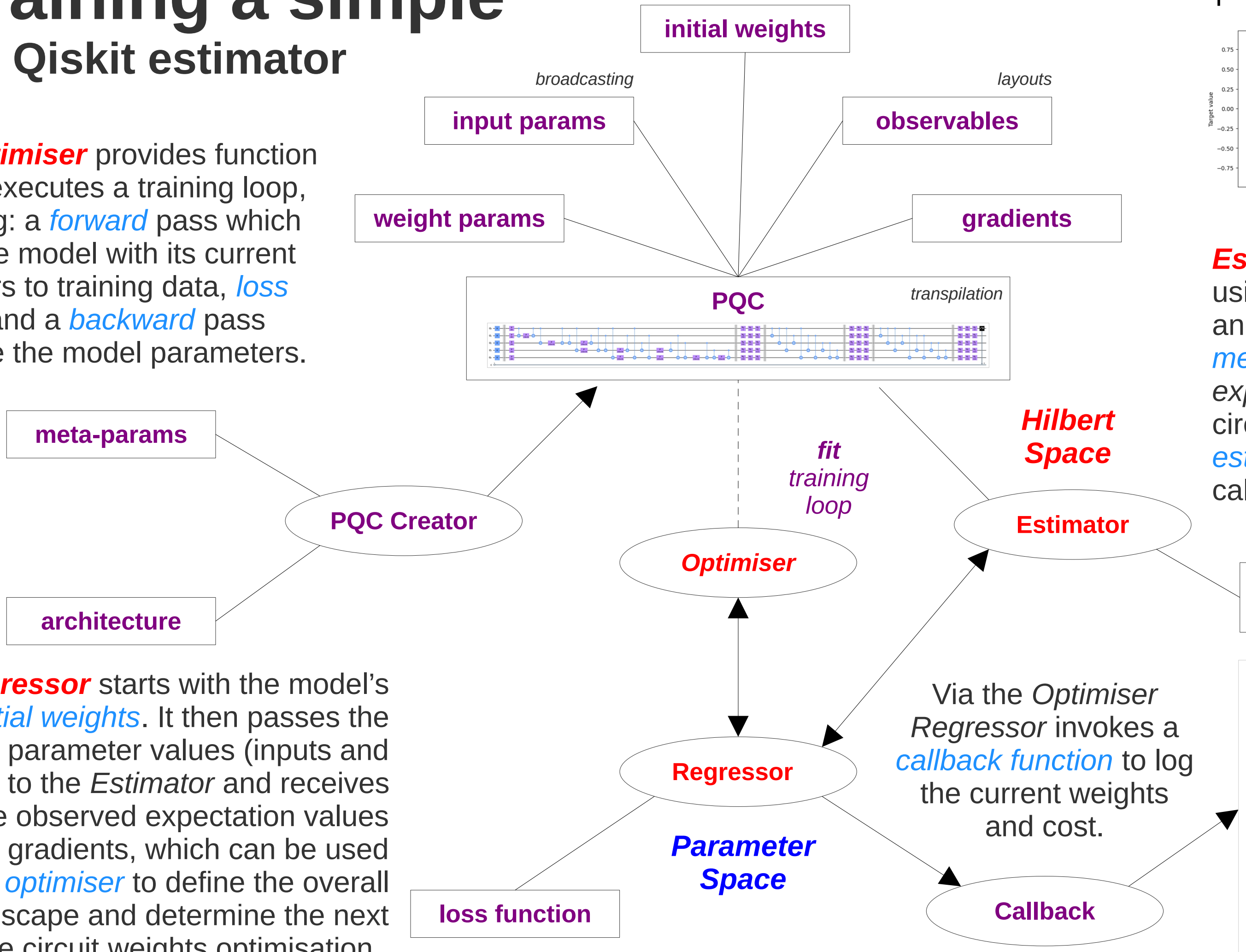
- use fewer qubits / layers / parameters
- use local cost functions (do not measure all qubits)
- use non-Euclidean metrics (e.g. Fisher Information Metric)
- beware of random params initialisation (and keep them small)
- use BP-resistant model design (e.g. layer-by-layer dev)
- use BP-resistant models (e.g. QCNNs)

Training a simple
TS Qiskit estimator

Dataset is to be prepared, cleaned and partitioned for training and testing.



Qiskit ***Optimiser*** provides function ***fit*** which executes a training loop, performing: a *forward* pass which applies the model with its current parameters to training data, *loss function*, and a *backward* pass to improve the model parameters.



Via the *Optimiser Regressor* invokes a *callback function* to log the current weights and cost.

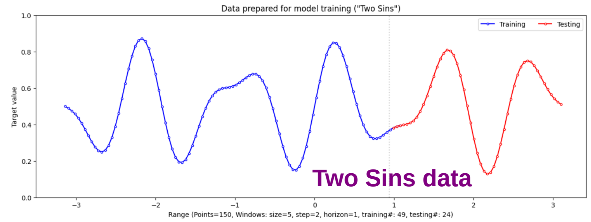
Estimator creates the physical circuit using the *observables*, *input parameters* and *weight parameters*, and the *gradient method used in the calculation of expectation values.* It then executes the circuit by relying on a hardware specific *estimator primitive*. It returns the calculated expectation values.

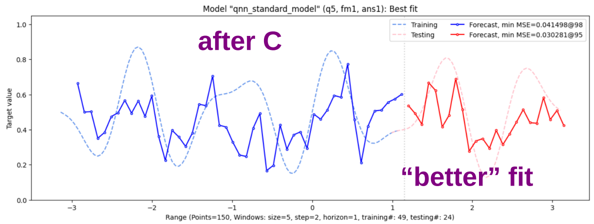
Regressor starts with the model's *initial weights*. It then passes the current parameter values (inputs and weights) to the *Estimator* and receives back the observed expectation values and their gradients, which can be used by an *optimiser* to define the overall cost landscape and determine the next step in the circuit weights optimisation.

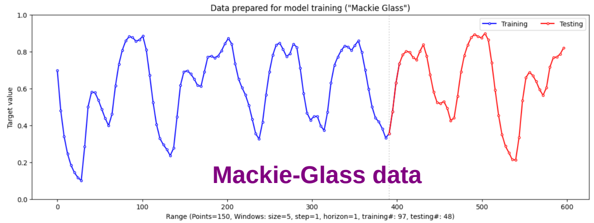
training log

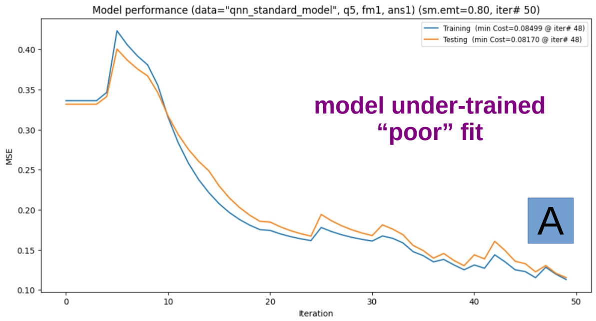
```
Model training started

(00:00:00) - Iter#:   0 / 500, Cost: 0.238564
(00:00:07) - Iter#:  50 / 500, Cost: 0.162685
(00:00:14) - Iter#: 100 / 500, Cost: 0.126066
(00:00:21) - Iter#: 150 / 500, Cost: 0.073866
(00:00:29) - Iter#: 200 / 500, Cost: 0.053152
(00:00:36) - Iter#: 250 / 500, Cost: 0.038513
(00:00:43) - Iter#: 300 / 500, Cost: 0.033054
(00:00:50) - Iter#: 350 / 500, Cost: 0.029146
(00:00:58) - Iter#: 400 / 500, Cost: 0.027865
(00:01:05) - Iter#: 450 / 500, Cost: 0.026759

Total time 00:01:12, min Cost=0.026013
```

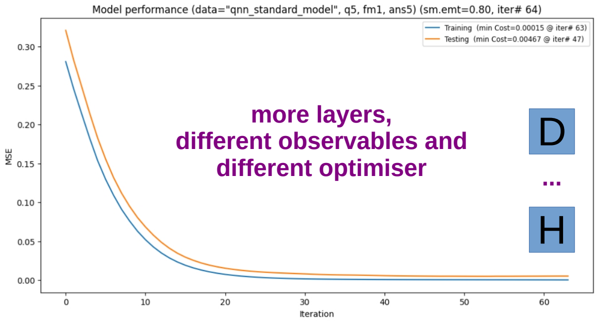
In search of a solution!

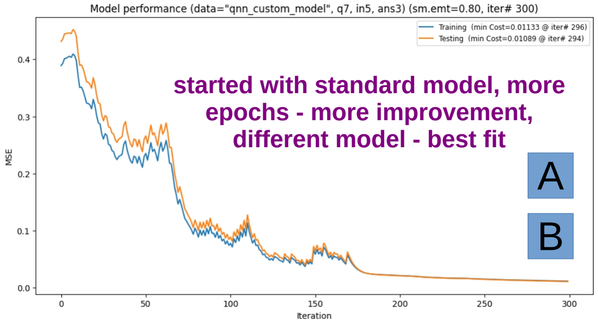


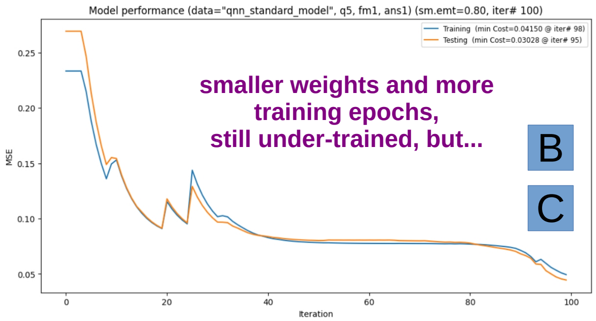


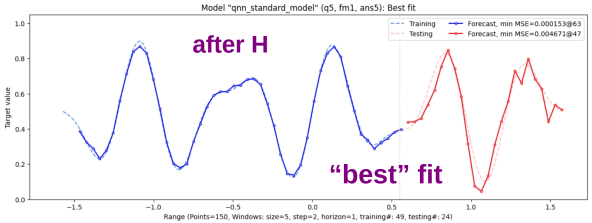


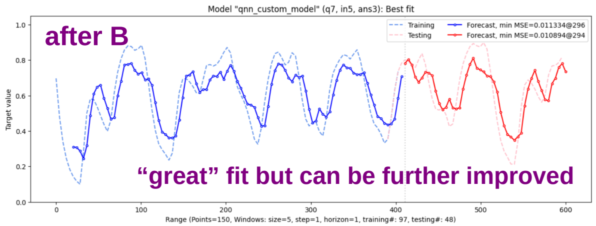












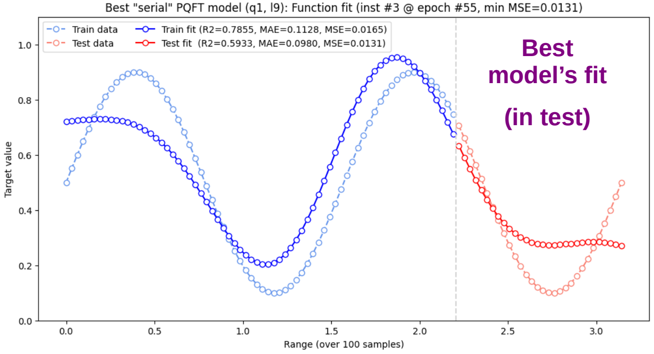
Change: new dataset using the standard and custom QNN model

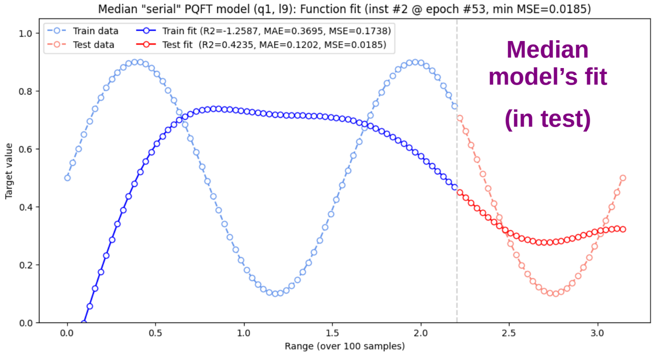
Often incremental changes result only in small improvements, we may need a drastic change: a model, optimiser or observables

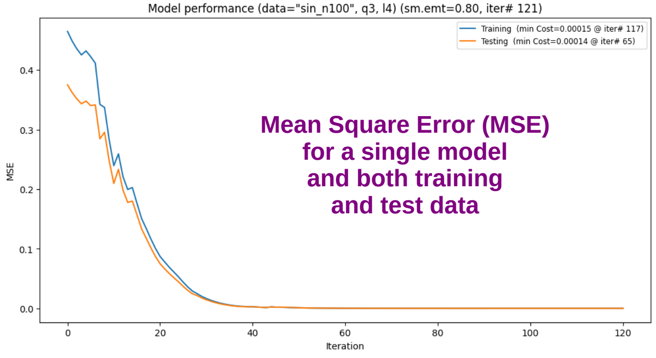
Task: improve a forecasting model for two datasets

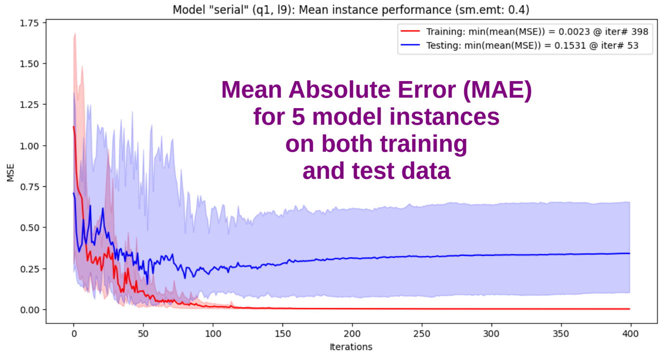
Quantum model performance:
Scoring a quantum model

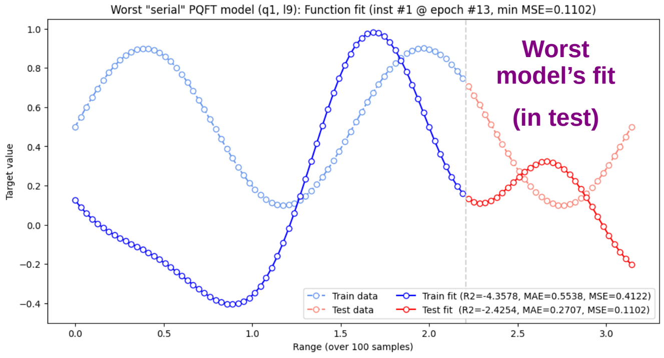
- Model training involves an optimizer, training data and a loss function, e.g. L2Loss (MSE).
- However, *several metrics may be needed to assess the model performance*, e.g. MSE, MAE or R^2, *to be calculated for training, validation and test data partitions*.
- At each optimisation step, the *model parameters should be saved for model scoring* on all data partitions (e.g. figure bottom-left).
- However, *quantum models are highly sensitive to their parameters initialisation*, therefore *performance of a single model run is not reliable*!
- So, we should *run multiple, differently initialised, instances of the same model* and analyse a distribution of their performance results.
- Here we present several (5) instances of the same model identically configured but differently initialised (figure bottom-middle).
- Set the model performance expectations by *indicating the model's fit to data*, depending on it best, median and worst instance performance (figures right).





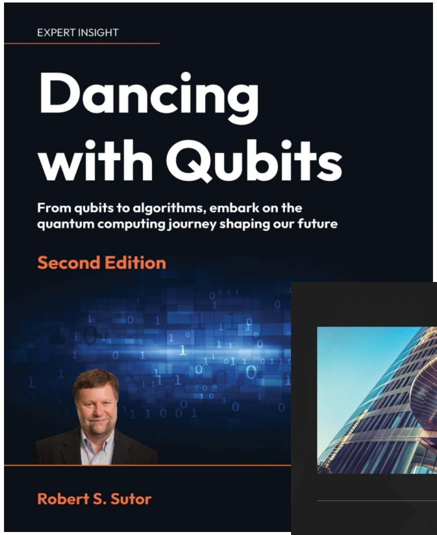


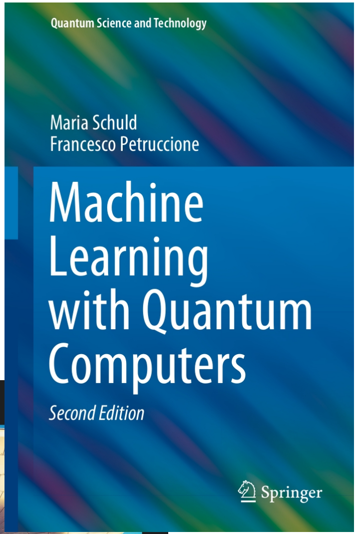


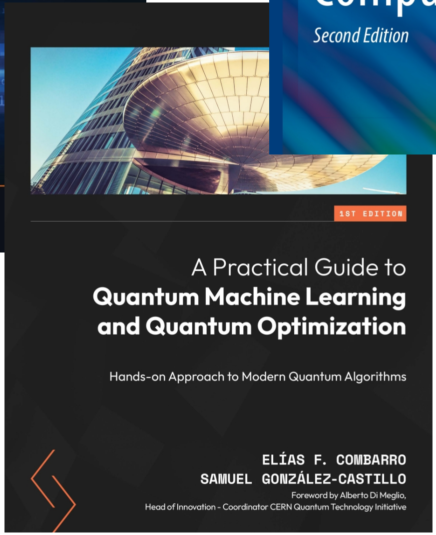


Recommended reading

on QML with Qiskit







Quantum computing with Qiskit

Ali Javadi-Abhari,[1] Matthew Treinish,[1] Kevin Krsulich,[1] Christopher J. Wood,[1] Jake Lishman,[2] Julien Gacon,[3] Simon Martiel,[4] Paul D. Nation,[1] Lev S. Bishop,[1] Andrew W. Cross,[1] Blake R. Johnson,[1] and Jay M. Gambetta[1]

[1]IBM Quantum, IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598
[2]IBM Quantum, IBM Research Europe, Hursley, United Kingdom
[3]IBM Quantum, IBM Research Europe, Zürich, Switzerland
[4]IBM Quantum, IBM France Lab, Orsay, France

We describe Qiskit, a software development kit for quantum information science. We discuss the key design decisions that have shaped its development, and examine the software architecture and its core components. We demonstrate an end-to-end workflow for solving a problem in condensed matter physics on a quantum computer that serves to highlight some of Qiskit's capabilities, for example the representation and optimization of circuits at various abstraction levels, its scalability and retargetability to new gates, and the use of quantum-classical computations via dynamic circuits. Lastly, we discuss some of the ecosystem of tools and plugins that extend Qiskit for various tasks, and the future ahead.

I. INTRODUCTION

II. DESIGN PHILOSOPHY

Qiskit Machine Learning: an open-source library for quantum machine learning tasks at scale on quantum hardware and classical simulators

M. Emre Sahin,[1] Edoardo Altamura,[1] Oscar Wallis,[1] Stephen P. Wood,[2] Anton Dekusar,[3] Declan A. Millar,[4] Takashi Imamichi,[5] Atsushi Matsuo,[5] Stefano Mensa,[1,*] and Code contributors

[1]The Hartree Centre, STFC, Sci-Tech Daresbury, Warrington, WA4 4AD, United Kingdom
[2]IBM Quantum, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
[3]IBM Quantum, IBM Research Europe – Dublin, Ireland
[4]IBM Research – UK
[5]IBM Quantum, IBM Research – Tokyo, Tokyo 103-8510, Japan
(Dated: Friday 13th June, 2025)

We present Qiskit Machine Learning (ML), a high-level Python library that combines elements of quantum computing with traditional machine learning. The API abstracts Qiskit's primitives to facilitate interactions with classical simulators and quantum hardware. Qiskit ML started as a proof-of-concept code in 2019 and has since been developed to be a modular, intuitive tool for non-specialist users while allowing extensibility and fine-tuning controls for quantum computational scientists and developers. The library is available as a public, open-source tool and is distributed under the Apache version 2.0 license.

I. INTRODUCTION

arXiv:2505.17756v1 [quant-ph] 23 May 2025

Qiskit SDK 2.2 release notes

2.2.1

Prelude

Qiskit 2.2.1 is a small patch release that fixes several bugs identified in the 2.2.0 release.

Qiskit 2.2.3 is a patch release with bug fixes

Summary and thank you!

- QML is an intersection of QC x ML x Maths
- Qiskit provides an excellent platform for QML
- Qiskit QML models are based in PQCs
- The most common approach to QML are VQAs
- Quantum encoding is the key to success (but full of traps)
- Qiskit provides tools and templates for ansatz design
- Measurement of circuits requires interpretation of results
- Quantum circuit design needs to consider its state evolution in Hilbert space and its parameter optimisation in classical parameter space, both have conflicting requirements
- Dimensionality of Hilbert space and parameter space promotes expressivity of the circuit, however, it hampers the model trainability
- Qiskit provides powerful runtime framework for training classification (sampling) and estimation models, equipped with noise suppression and mitigation tools
- Quantum models are highly sensitive to initialisation, so their performance needs to be assessed across different model instances
- QML is still a research discipline
- Adapting ML methods to QML has not shown an advantage
- The advantage of QML over ML can only be found in Hilbert space

Any questions?

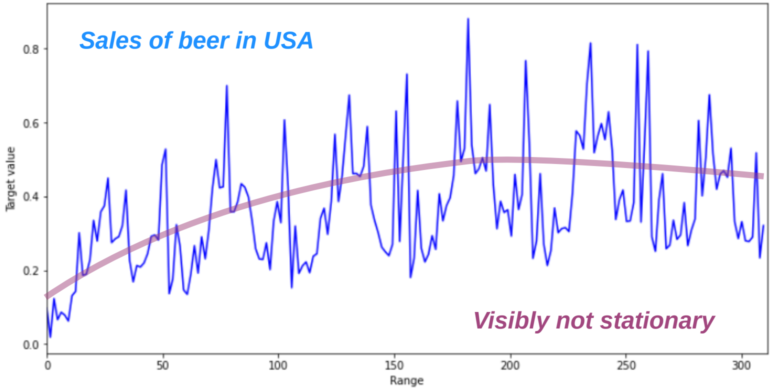
Available resources, see:
ironfrown (Jacob L. Cybulski, Enquanted)
https://github.com/ironfrown/qml_bcd_lab

Images from Unsplash and Wikipedia
Enquanted is being somewhere in-between Enchanted and Entangled

Example:
quantum time series analysis

Quantum time series analysis is hard!

- Time series (TS) analysis aims to *identify patterns* in historical time data and to *create forecasts* of what data is likely to be collected in the future
- *Many TS applications*, including heart monitoring, weather forecasts, machine condition monitoring, etc.
- Time series can be *univariate* or *multivariate*
- Time series often show *seasonality* in data, i.e. some patterns repeating over time



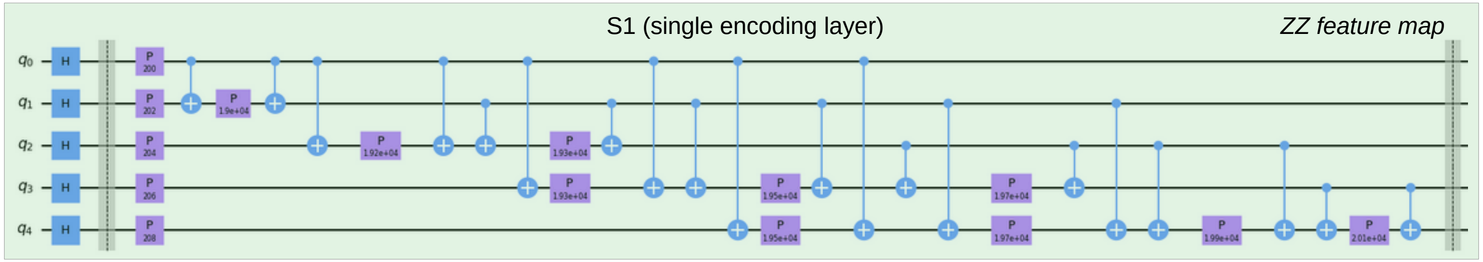
Mean is not constant

Variance is not constant

Trend is non-linear

- TS values are dependent on the preceding values!
- Distinction between consecutive TS values is small!
- There are several different types of TS models, e.g.
  - The first group are *curve-fitting models*, which are trained to fit a function to a sample of data points, to predict data values at specific points in time
  - The second group are *forecasting models*, which are trained to predict future data points from their preceding temporal context (a fixed-size window sliding over TS)
- Majority of statistical forecasting methods require *strict data preparation*, such as dimensionality reduction, TS aggregation, imputation of missing values, removal of noise and outliers, adherence to normality and homoskedasticity, they need to be stationary
- QML methods do not have such strict requirements, and are promising for effective time series analysis and forecasting!

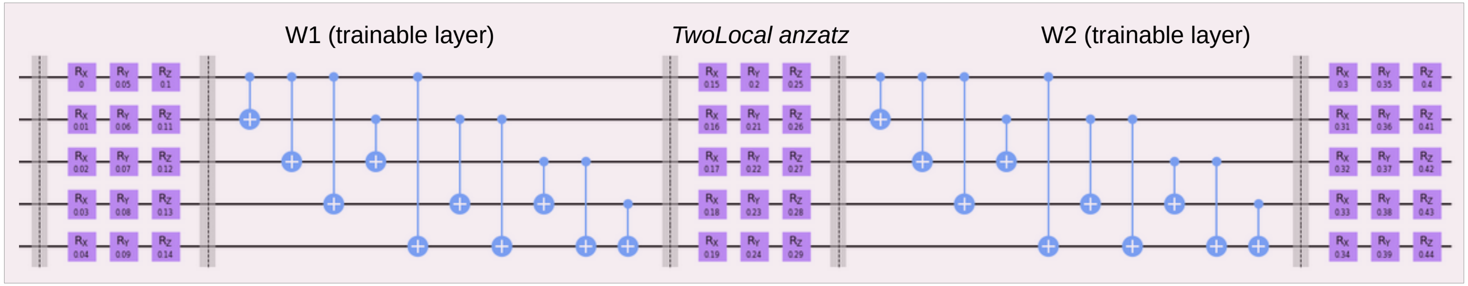
Forecasting models

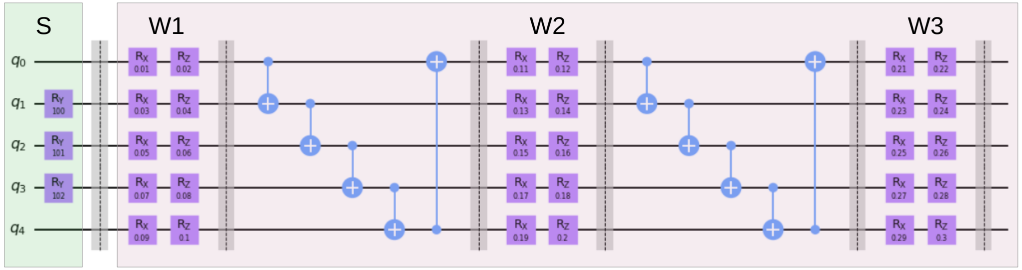


xqnn_standard_model

This QNN model consists of two components, i.e. (1) the ZZ feature map, and (2) a TwoLocal ansatz.

This QNN architecture is commonly used and considered to enhance expressivity of the quantum circuit.





xqnn_custom_model

This is a custom-built QNN model. It is composed of (1) an angle encoding feature map of Ry rotation gates; and (2) an ansatz that is wider than the encoding layers and consisting of several trainable layers of Rx, Ry and Rz parameterised blocks interspersed with entangling blocks of CNOT gates arranged in a circular fashion.

Custom models are often used when facing training difficulties, e.g. to improve the circuit trainability by reducing the its entanglement (fewer CNOT gates) or to add trainable parameters to enhance its expressivity.

In this workshop we provide two alternative QNN models. The first features the commonly used circuit structure relying on Qiskit supplied parameterised circuits. The second is custom made and is created from the Qiskit basic building blocks (gates and parameters).