



Secrets revealed in this session:

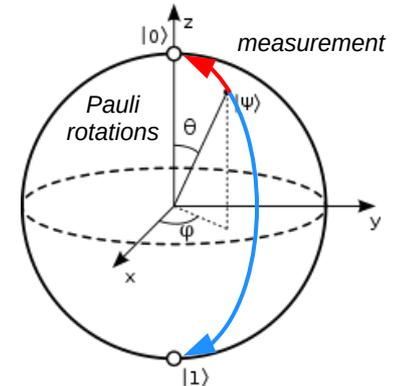
To explore and explain QML and QAI in about less than a blink of an eye!

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

See: [Ironfrown \(Github\)](#)

Quantum AI and ML

Jacob L. Cybulski
Enquanted, Australia
and Deakin University, SIT

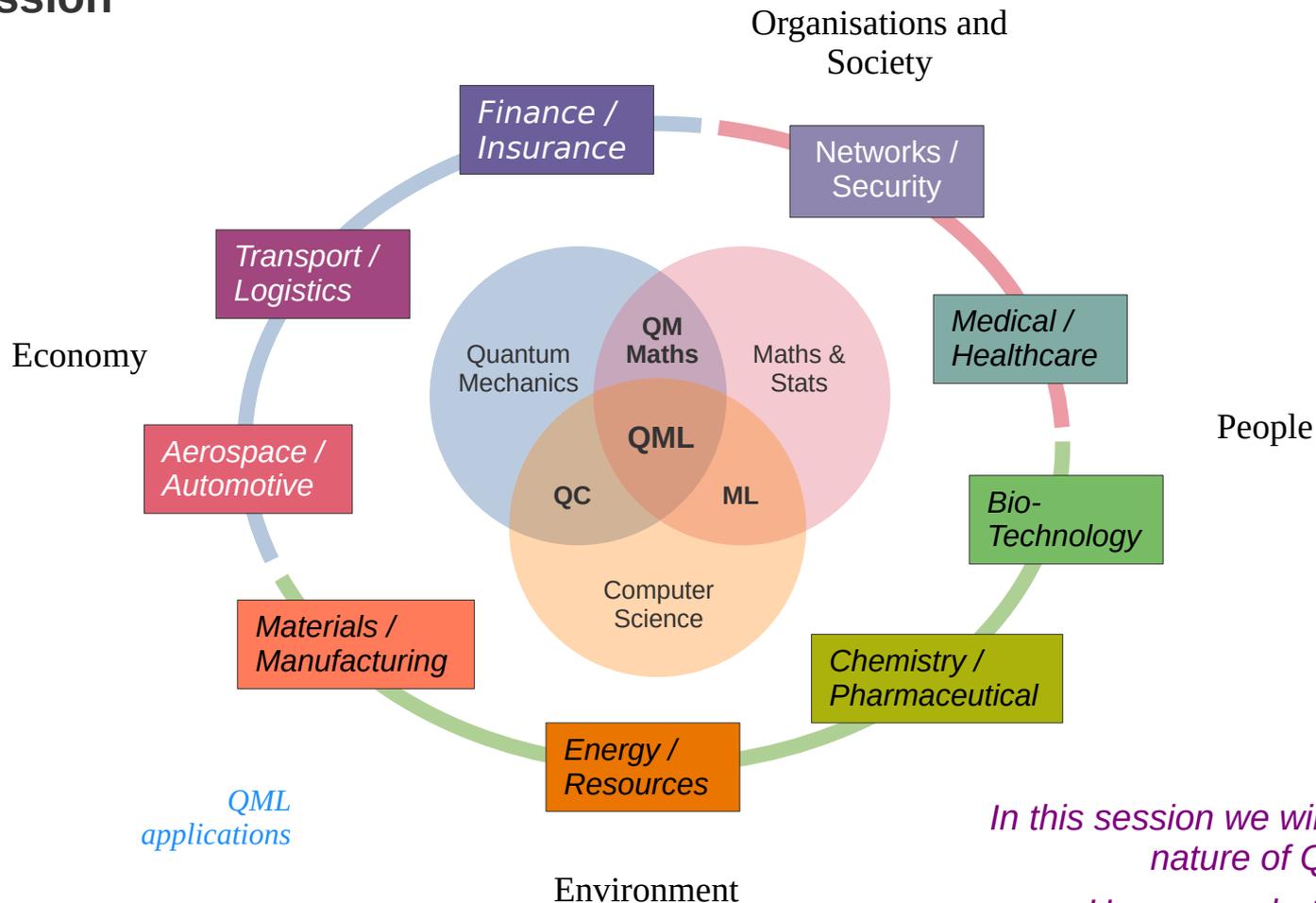


Quantum ML

aims of this session



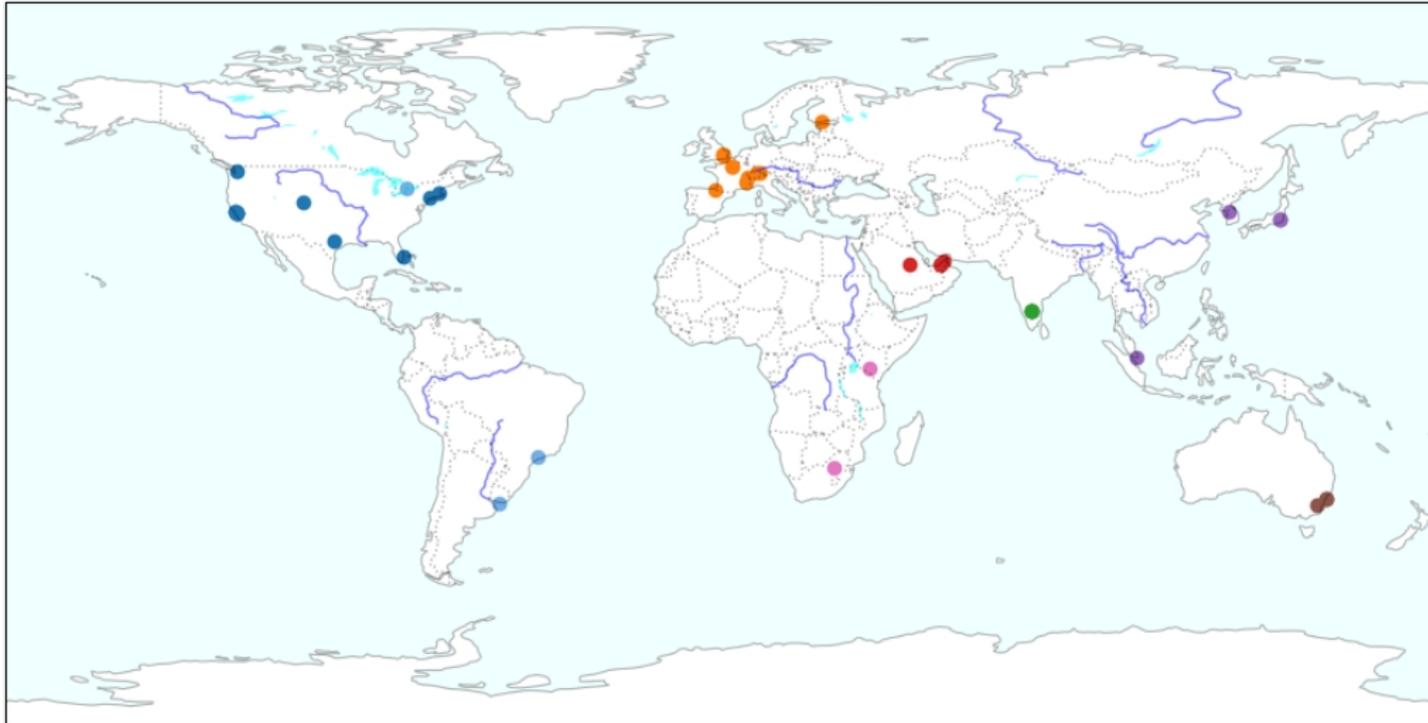
Jacob Cybulski, Founder
Enquanted, Australia



*In this session we will explain the nature of QML models.
However, what is a model?*

Where are the main QML / QAI hubs?

Location of the main QML companies



Quantum-Native QML

designed for true quantum execution

- PennyLane (Xanadu)
- Qiskit ML (IBM)
- Pulser/Cadence (Pasqal)
- CUDA Quantum (NVIDIA)

Kits with 3rd Party QML

also designed for true quantum execution

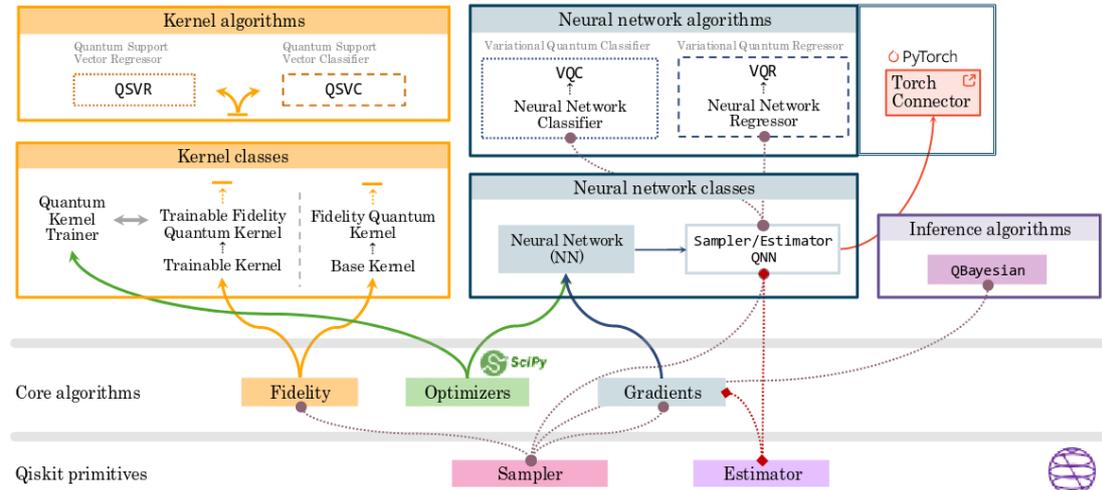
- Bloqade with PennyLane (QuEra)
- TensorFlow Quantum (Google)
- Braket SDK/PL&TF (AWS)
- Azure Quantum/QDK (Microsoft)

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)

Classical MLP

Multi-Layer Perceptron

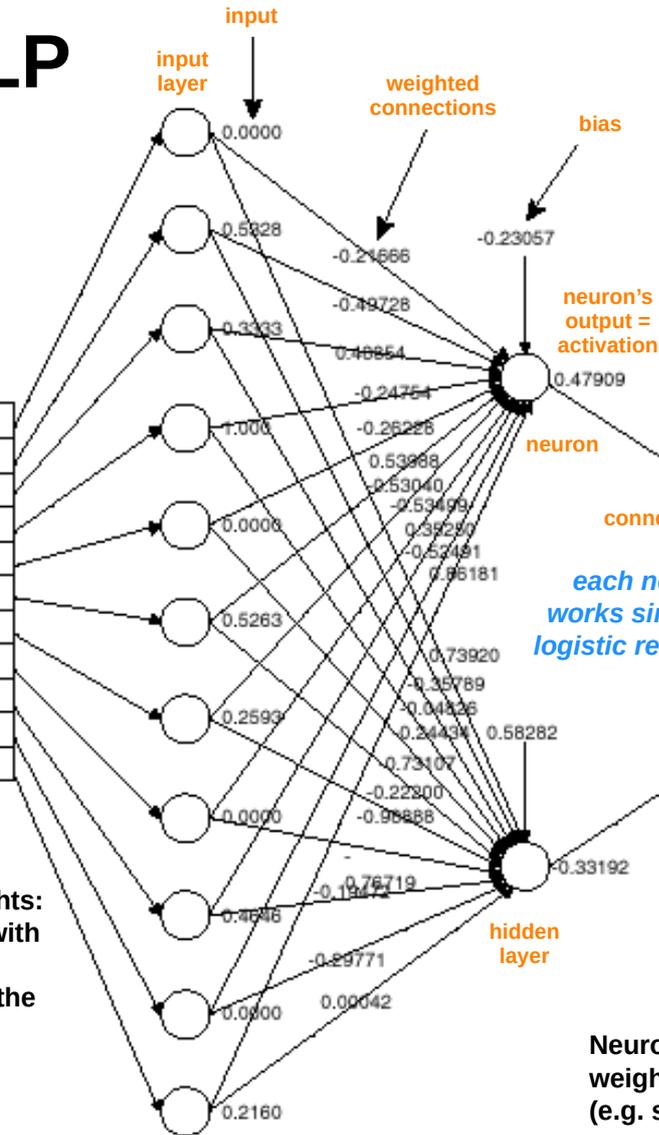
Input layer is often responsible for the **normalization** of input values into the range of [-1..1] (or [0..1])

Num_Apartments	1	0.0000
Year_Built	1923	0.5328
Plumbing_Fixtures	9	0.3333
Heating_Type	B	1.0000
Basement_Garage	0	0.0000
Attached_Garage	120	0.5263
Living_Area	1614	0.2593
Deck_Area	0	0.0000
Porch_Area	210	0.4646
Recroom_Area	0	0.0000
Basement_Area	175	0.2160
Price	176211	

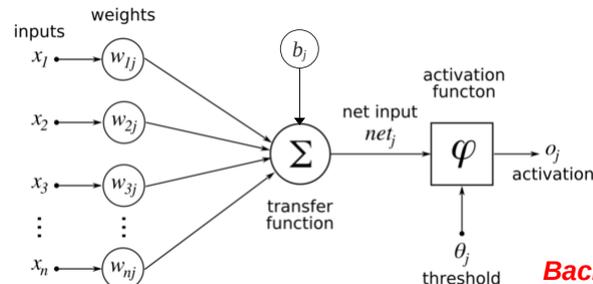
a single data record

Gradient-based optimizer repeats the following steps to optimize the model weights:

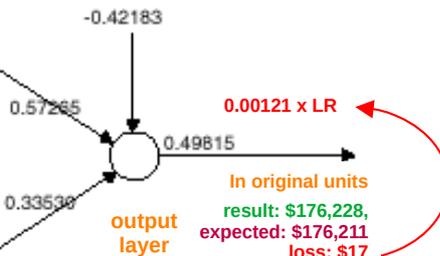
- **forward pass** which applies the model with its current weights to the training data;
- calculation of the result and error from the expected value using a **loss function**;
- **backward pass** to improve the model weights to reduce the error.



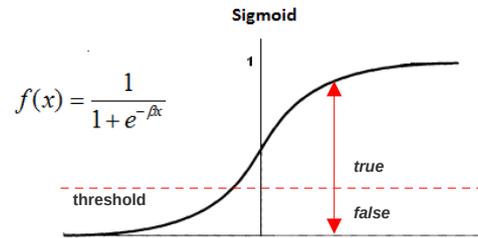
Each hidden layer neuron uses a **transfer** and **activation** functions to calculate the unit's output value based on inputs and weights, **threshold** is used when the final output is used for classification (e.g. binary).



Backpropagation improving weights by propagating errors back from output towards input, minimizing loss iteratively by adjusting weights a fraction of the error at each cycle.



The process is slow but can be parallelized by using efficient matrix calculations



Neuron values must be within a range -1..1, aggregation of weighted inputs is transformed by the **activation function** (e.g. sigmoid or logistic).

Optimisation example

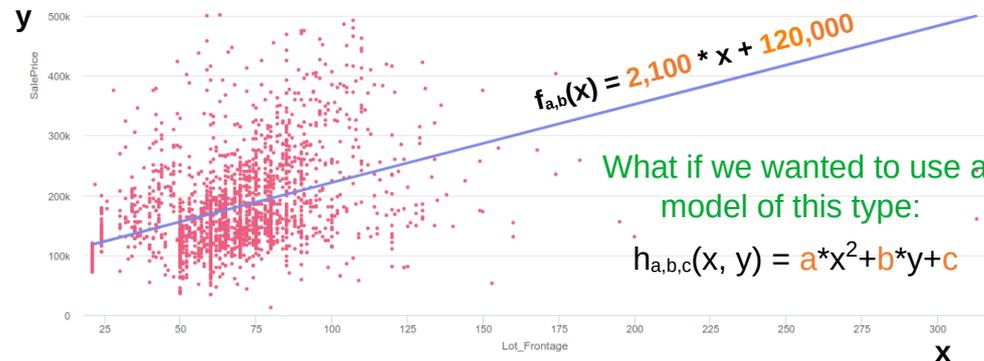
Gradient descent

Consider the house price (y) as a function of the size of its front yard (x). Let us consider all **models** ($f_{a,b}$) to estimate the house prices by the formula:

$$y = f_{a,b}(x) = a * x + b$$

Each model is parameterized by weights a and b , and can fit a sample $A=\{x, y\}$ of house training data (here we used Ames real estate dataset).

For each house (x, y), a model $f_{a,b}$ will make some error (**loss**). For all houses in A it will accumulate these errors as a single value (**cost**), e.g. **MAE** (mean absolute error).



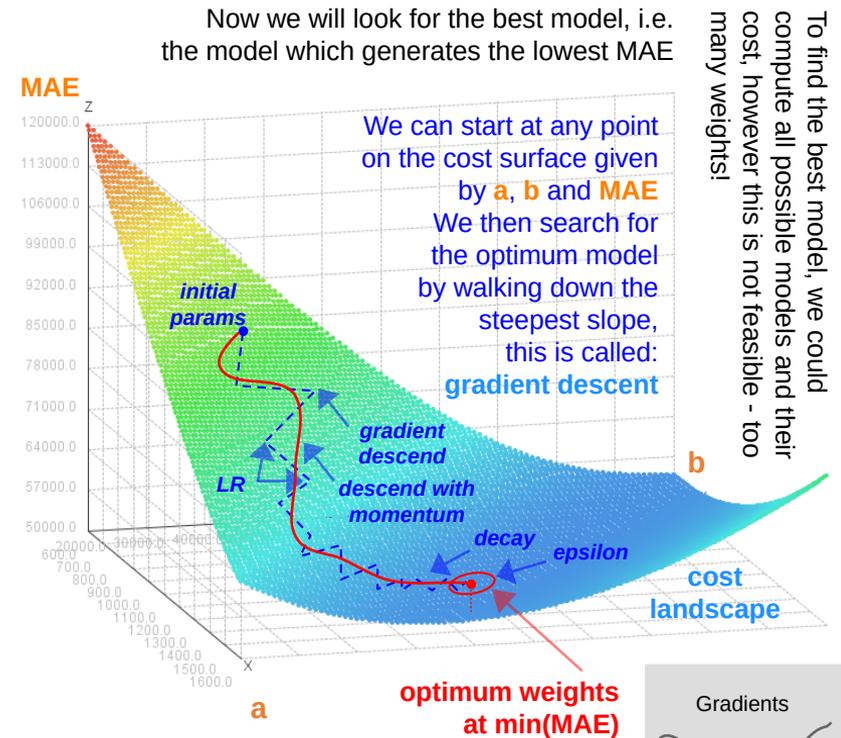
The cost of each model is a point in a 3D space

$$a \times b \times \text{MAE}$$

All such points form a "cost" surface.

The shape of such a surface we call the **cost landscape**.

When a model has many weights, the cost surface is multi-dimensional and called a manifold.



The optimizer controls this process via its hyper-parameters, i.e. parameters of the gradient descent itself:

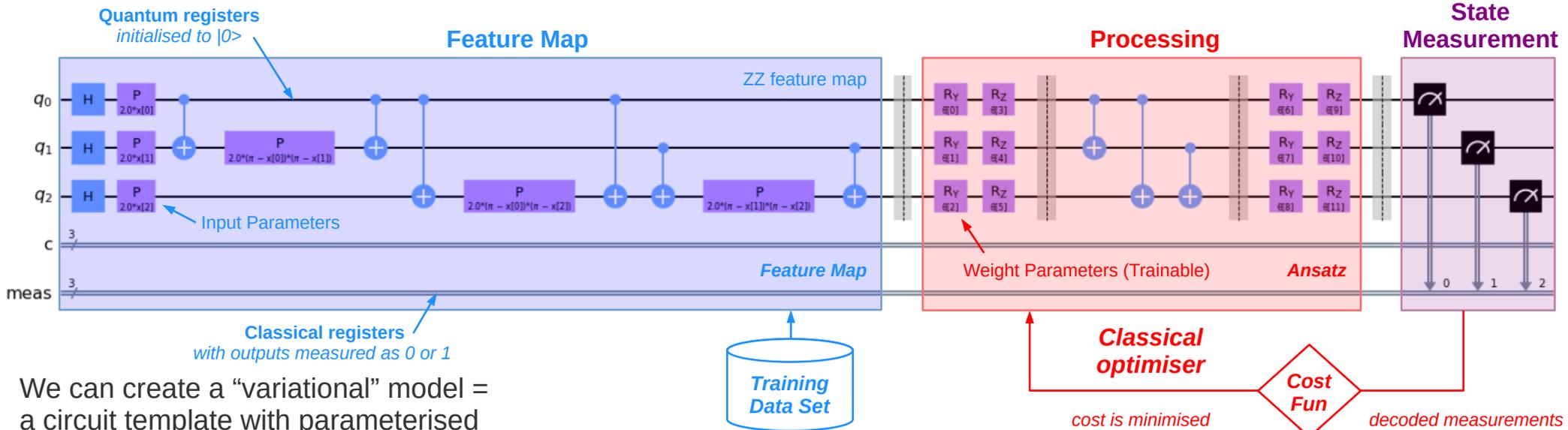
learning rate
momentum
decay
epsilon

By using gradient descent, the optimum cost (and thus the model), was found at:

A=1060 (Lot_Frontage)
B=90000 (Intercept)
MAE=53473.097 (Error)

Parameterized Quantum Circuits (PQC) & Variational Quantum Algorithms (VQA)

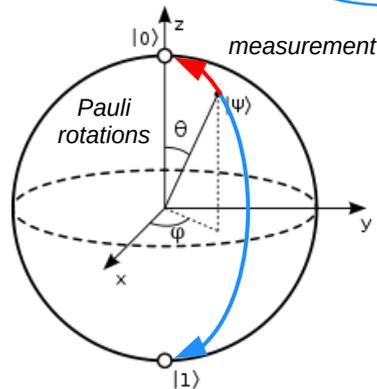
*Variational quantum circuits are not executable!
Their input and weight params must be assigned values!
Backpropagation cannot work on quantum machines!*



We can create a “variational” model = a circuit template with parameterised gates, e.g. $P(a)$, $R_y(a)$ or $R_z(a)$, 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.

cost is minimised during circuit training

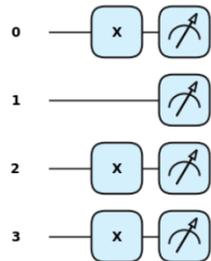
decoded measurements are matched against training data

Data Encoding

Many encoding methods, e.g. basis, angle, amplitude, QRAM, ...

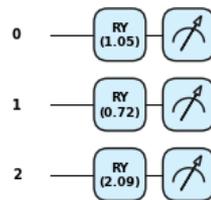
Maria Schuld and Francesco Petruccione
Machine Learning with Quantum Computers.
2nd ed. Springer, 2021.

Basis encoding



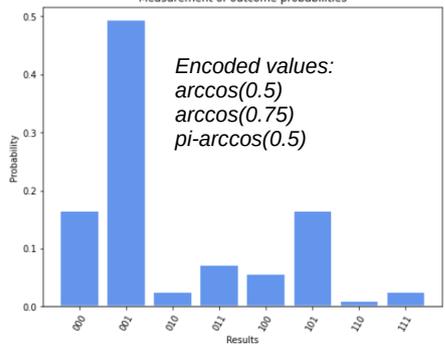
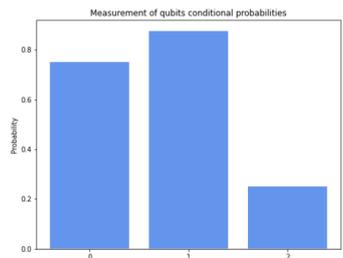
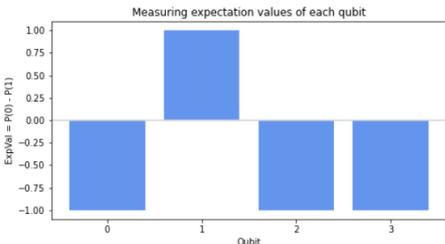
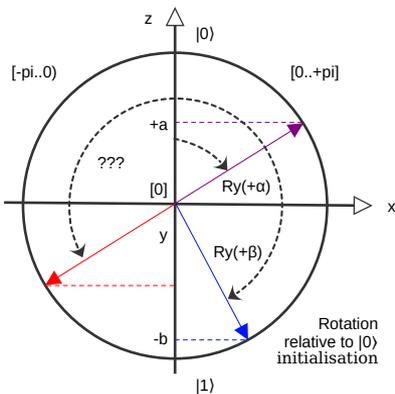
The simplest data encoding and very popular, however, little data can be encoded at a time, you can encode only binary values per each qubit.

Angle encoding

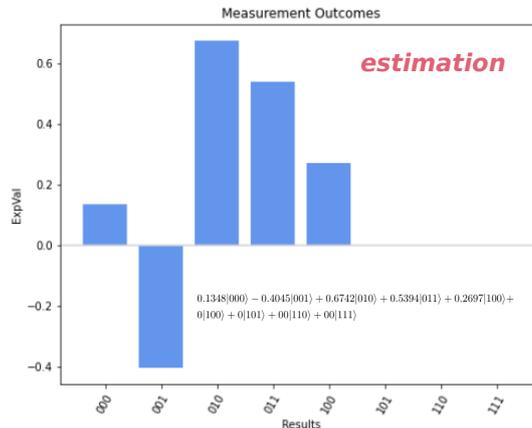
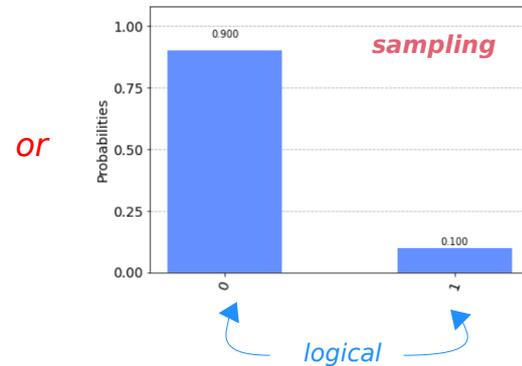
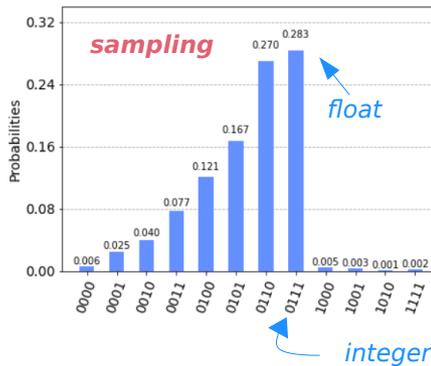
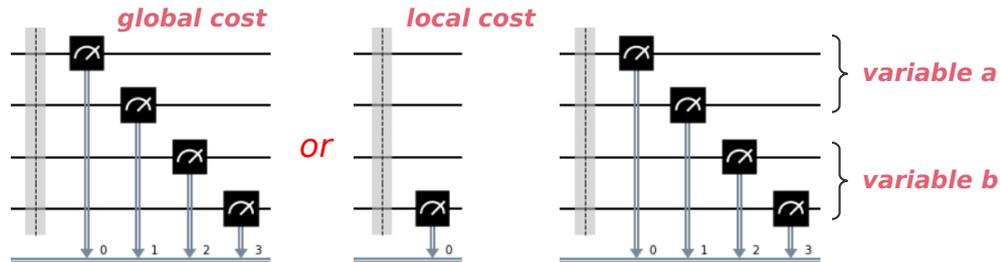


One of the most flexible and very effective, you can encode floating point numbers.

What you encode depends on your intention!



Encoding can be repeated across the circuit, which is called data reloading



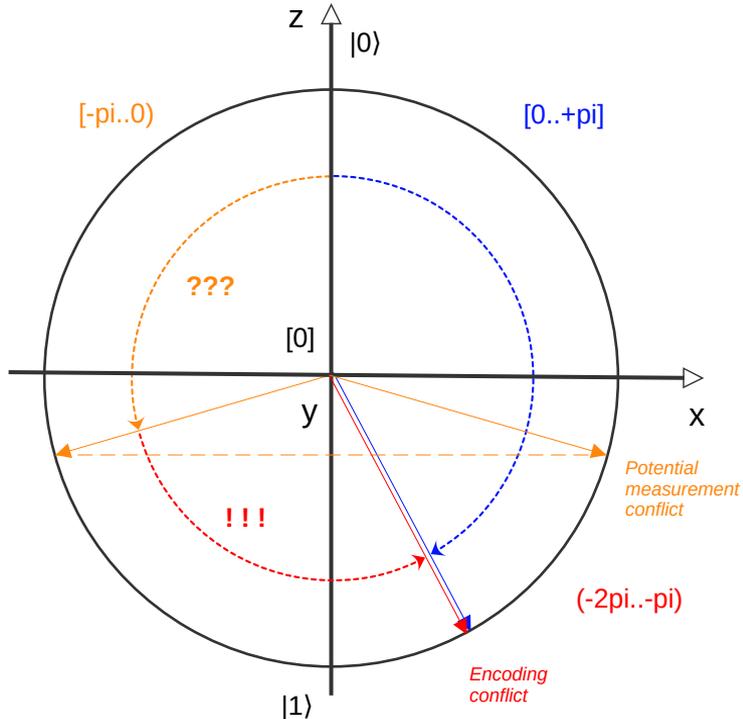
Measurements must be repeated, collected and then can be interpreted in many different ways

It is also possible to measure mid-circuit, however, beware as the circuit is no longer unitary and not reversible!

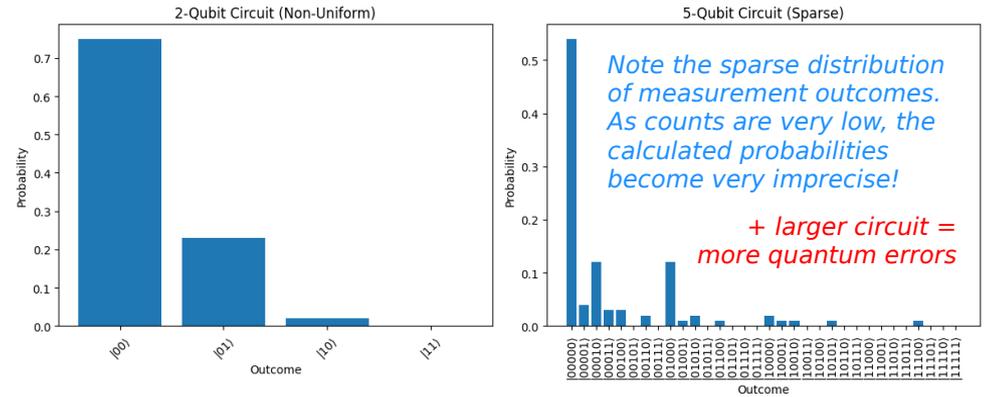
State Measurement

Encoding nightmares

- This example shows incorrect encoding which wraps its values around the Bloch sphere (several times).
- This results in different values to be mapped into the same amplitude (orange), which sometimes can be corrected by trainable rotational operations (Rxyz).
- This could also result in different values to be mapped into the same angles (red), which cannot be corrected.



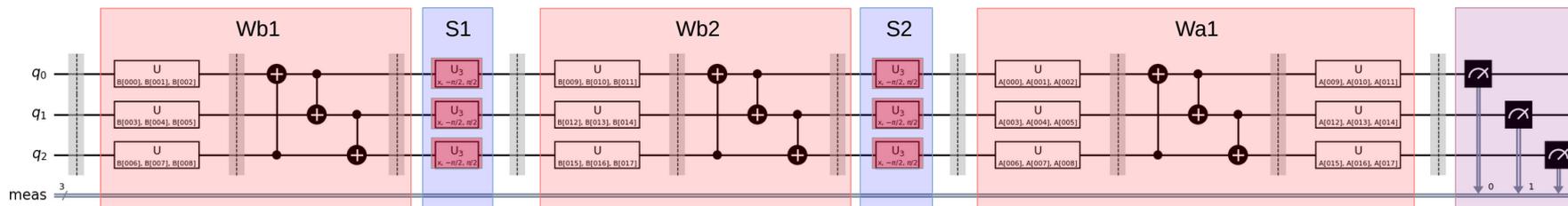
Measurement nightmares



- When increasing the number of measurements, we also exponentially increase the number of outcomes, which needs exponential increase of circuit runs!
- Unless the number of runs is increased with measurements, distribution of outcomes becomes sparse and probability calculations become imprecise.
- When working with large circuits, instead of measuring in terms of probability distributions we need to switch to expectation values, which scale with the required precision rather than numbers of qubits.
- At some point, measurements also suffer from quantum noise and it becomes necessary to deploy error suppression and mitigation.

Ansatz design

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



Beware that **adding qubits adds parameters and entanglements!**

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

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

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

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

$U(z,y,z)$

feature maps vary in:
structure and function

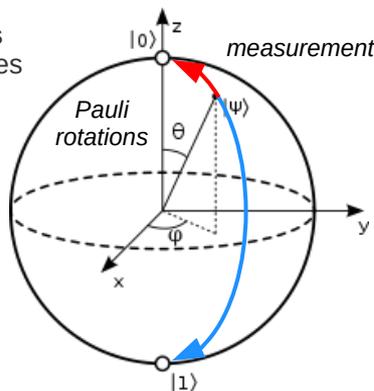
ansatze vary in:

- 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)

Expressivity vs Trainability

Parameter space (classical)

(dim = the number of params)

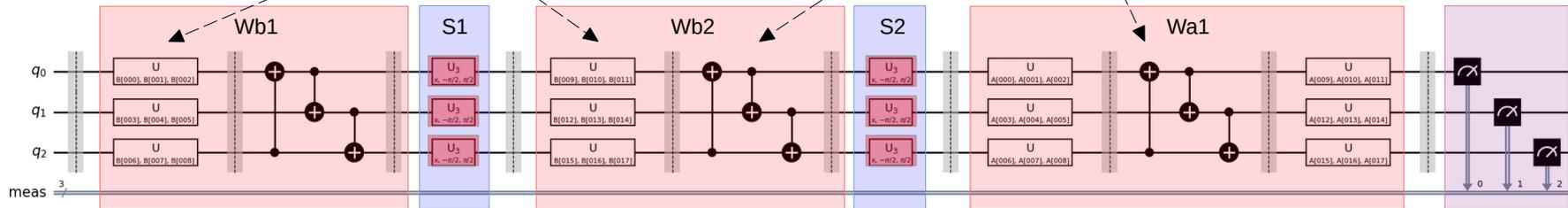
is a classical multi-dim space of trainable gate parameters, which the optimiser navigates – this is the only info available to the optimiser!

Data encoding

brings the 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 deform the Hilbert space geometry, and also the cost landscape used by the optimiser, entanglements cause non-local interactions



Hilbert state space (quantum)

(dim $\approx 2^{\text{the number of qubits}}$)

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)

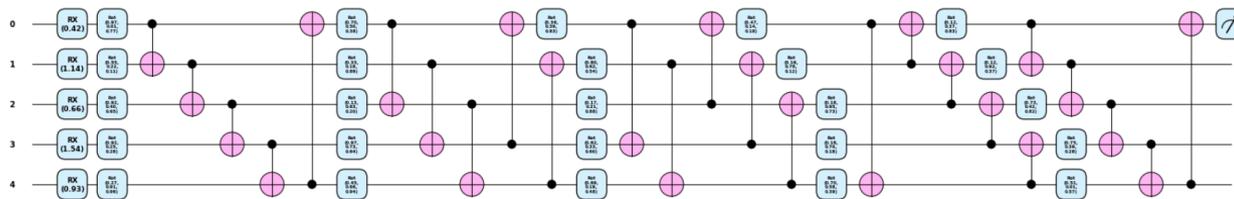
Sample Model Training:

Estimate diabetes progression one year after baseline

Quantum model training

Model training started

0	(000024 sec):	Loss 0.2452	R2 -3.0295
7	(000189 sec):	Loss 0.0971	R2 -0.5967
14	(000354 sec):	Loss 0.0596	R2 0.0204
21	(000519 sec):	Loss 0.0499	R2 0.1802
28	(000684 sec):	Loss 0.0455	R2 0.2517
35	(000848 sec):	Loss 0.0421	R2 0.3077
42	(001013 sec):	Loss 0.0404	R2 0.3354
49	(001178 sec):	Loss 0.0388	R2 0.3618
56	(001343 sec):	Loss 0.0385	R2 0.3669
63	(001507 sec):	Loss 0.0371	R2 0.3904
70	(001671 sec):	Loss 0.0359	R2 0.4102
77	(001835 sec):	Loss 0.0347	R2 0.4293
84	(002000 sec):	Loss 0.0349	R2 0.4261
91	(002164 sec):	Loss 0.0343	R2 0.4368
98	(002329 sec):	Loss 0.0329	R2 0.4586
105	(002493 sec):	Loss 0.0324	R2 0.4673
112	(002657 sec):	Loss 0.0333	R2 0.4525
119	(002822 sec):	Loss 0.0313	R2 0.4859
126	(002986 sec):	Loss 0.0312	R2 0.4870
133	(003151 sec):	Loss 0.0316	R2 0.4811
140	(003315 sec):	Loss 0.0321	R2 0.4727
147	(003479 sec):	Loss 0.0308	R2 0.4935



Which estimator is better?
Which could still improve?

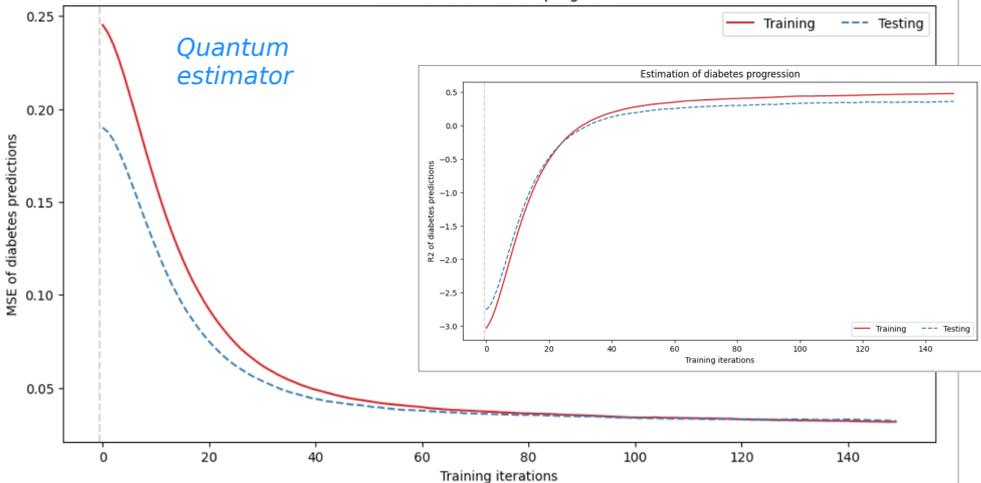
Would this change if we
were running the model
training on a quantum
machine?

devices = cpu + lightning.qubit
samples = 296, features = 5, params = 75, epochs = 150
training: cost = 0.0306 @ 0141, r2 = 0.4977 @ 0141
testing: cost = 0.0309 @ 0148, r2 = 0.3891 @ 0148
elapsed time = 3526sec (00:58:46)

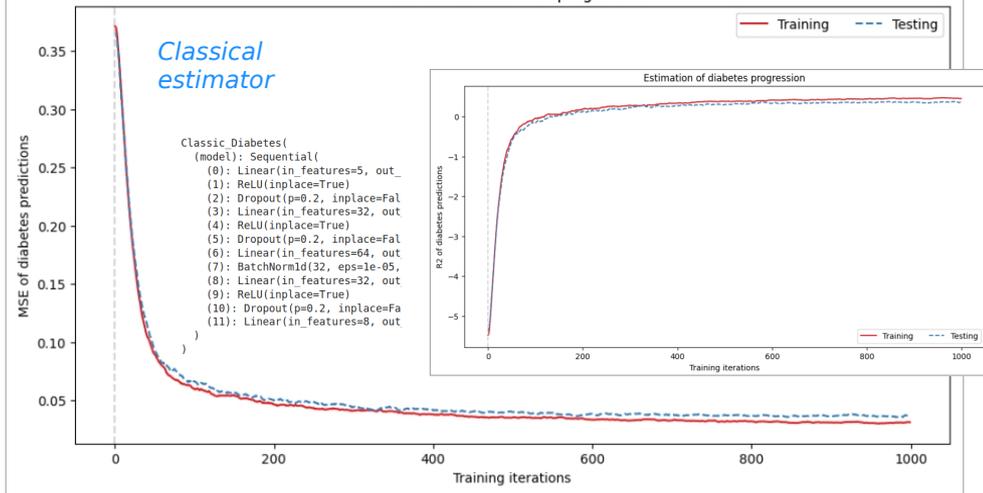
device = cpu
samples = 296, features = 5, params = 4721, epochs = 1000
training: cost = 0.0278 @ 0852, r2 = 0.5147 @ 0852
testing: cost = 0.0304 @ 0980, r2 = 0.4708 @ 0980
elapsed time = 3sec (00:00:03)

Total training time: 3526s (00:58:46)

Estimation of diabetes progression

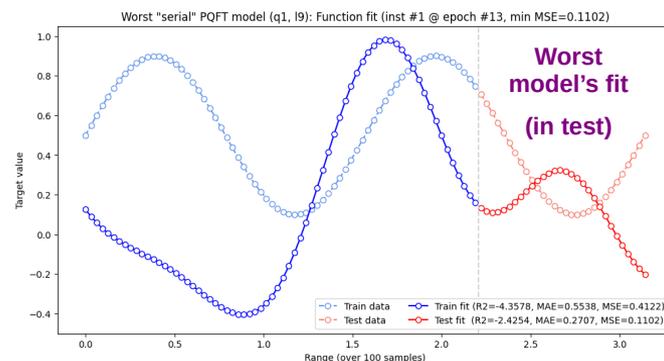
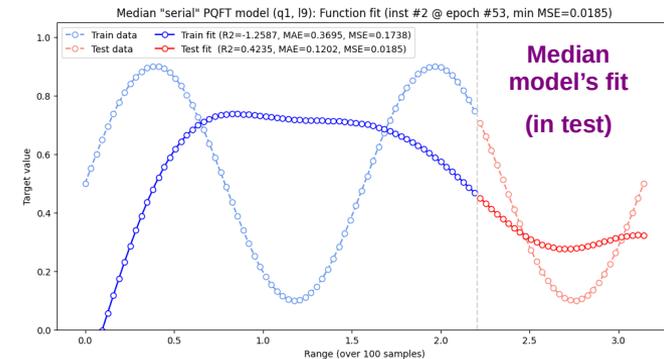
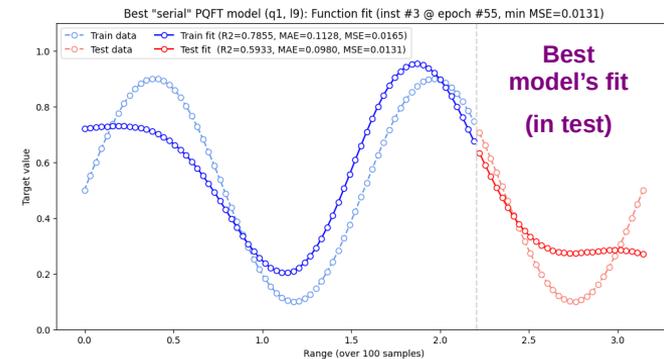
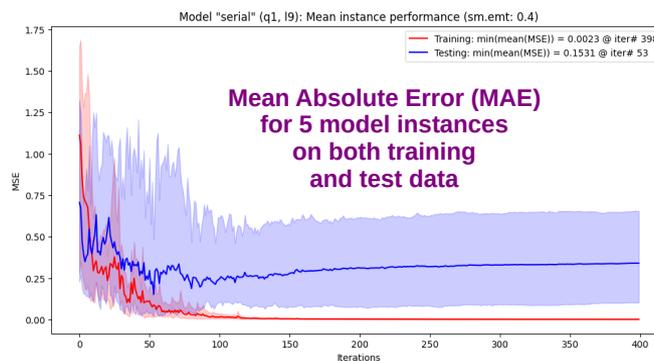
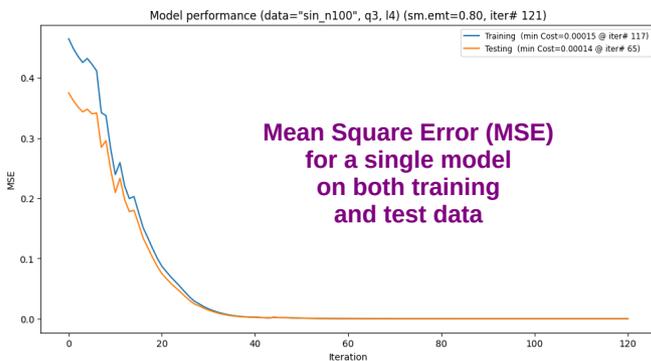


Estimation of diabetes progression



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).



Why we are getting high errors?

The reasons we are getting errors (residuals) ...

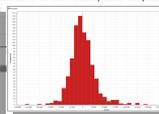
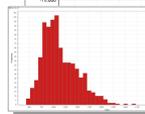
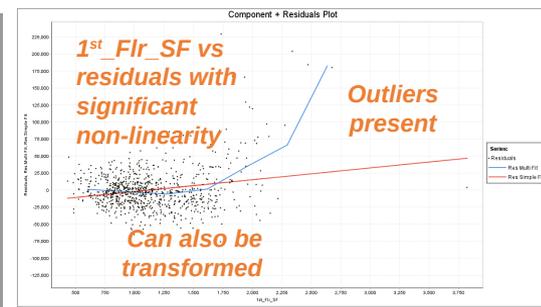
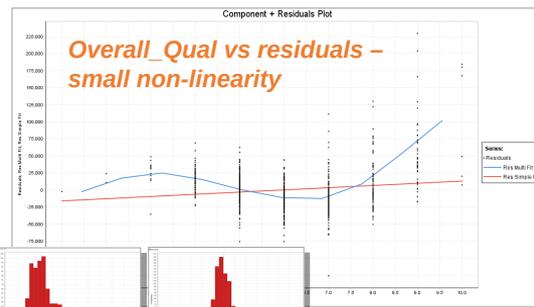
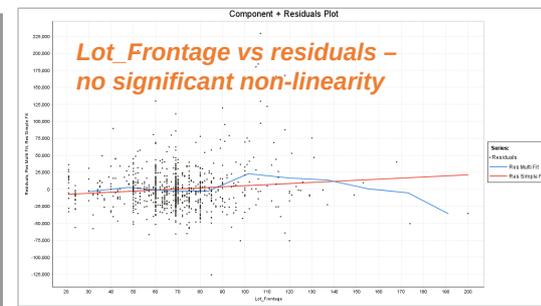
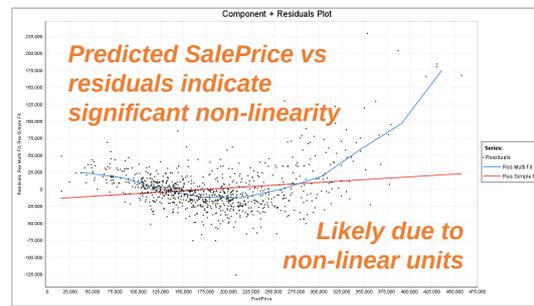
The quantum reasons:

- There are problems with the model/ansatz design
 - not expressive enough / too simple, e.g. too few params
 - too complex, e.g. too many qubits / layers / params
 - does not fit data, e.g. temporal / spacial data
- There are problems with model optimization
 - Barren plateaus, i.e. vanishing gradients
 - Under / over training, i.e. too few / too many epochs
 - Bad initialisation, e.g. random (far from optimum)
 - Poor data encoding / lack of reuploading
 - Poor observables / measurement strategy
 - Poorly selected optimiser / cost function
 - Continuous cost (MSE) / nominal score (e.g. accuracy)
 - We lose precision / phase on measurement

Classical / other reasons:

- Training data not representative (bad test results)
- Lack of cross-validation (you were lucky / or not)
- Poor data preparation (myth: prep not needed in Q)
- Presence of outliers / anomalies
- Other reasons

Understand your errors
- so, experiment and measure!



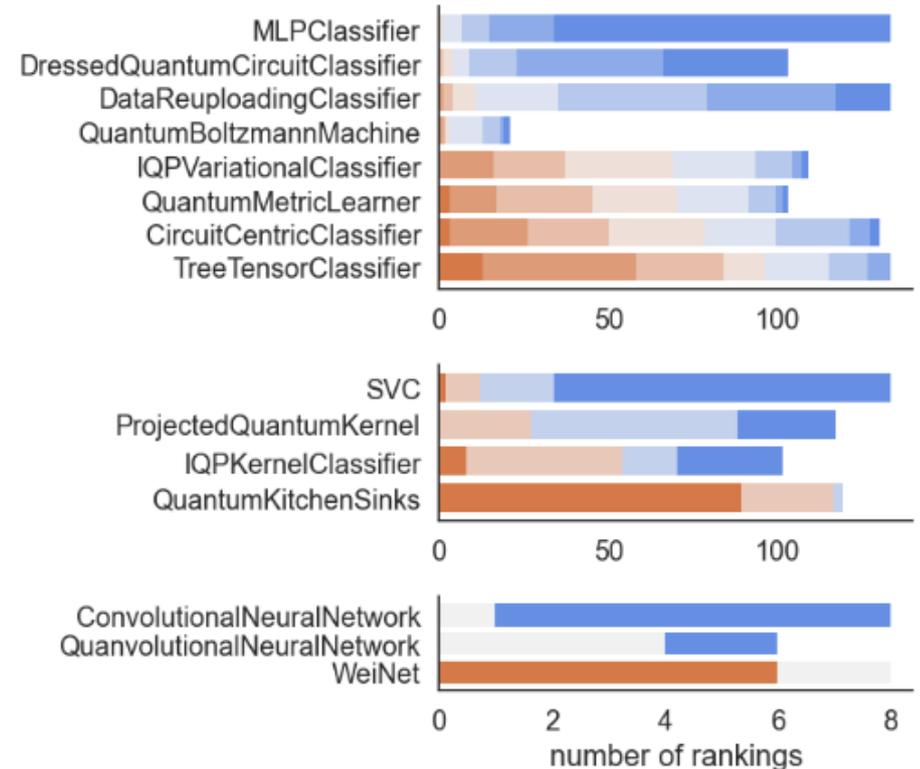
1st_Flr_SF

Residuals

Quantum vs Classical:

Will QML give an advantage?

- *Recent benchmarking show that classical models outperform quantum models (Bowles, et al, 2024)*
- Quantum advantage over classical models cannot be easily verified, and experiments cannot be reproduced!
- *Dressed models* (NNs with a quantum layer) perform well, yet it cannot be proven it is due to the quantum element
- *Data re-uploading* genuinely improves the quantum model's performance
- *Good fit between the model and data* has a huge impact on the performance far more than the classical models
- *Lessons learnt:*
 - when introducing a quantum method to machine learning, we need to carefully establish in what way this may alter or benefit the better established classical approaches
 - rather than adapting a classical model, we may need to introduce a unique quantum approach to model creation and optimisation!
- *QML is still in its early development - the new field is very exciting and very frustrating!*



(rankings: blue/best to red/worst)

What about QC+AI > QML?

Some definitions:

- **ML goals:** to perform through data and learning algorithms
- **AI goals:** to act intelligently (using ML)

Examples:

LLM	RL	Diffusion Models
Agentic AI	AlphaZero	Midjourney

Classical AI > ML where QC had little impact so far:

- Reasoning
- Planning
- Knowledge representation
- Causal inference
- NLP

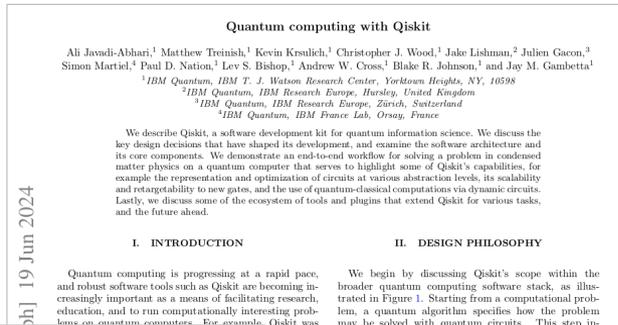
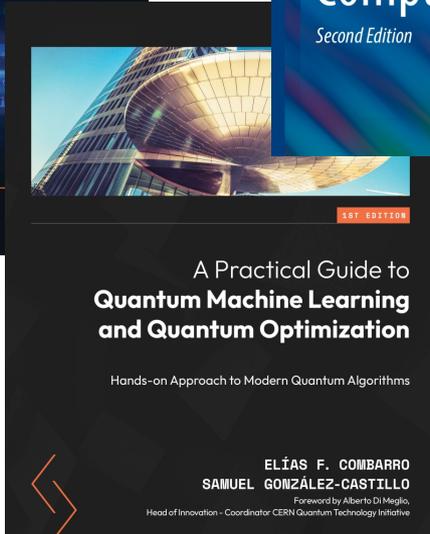
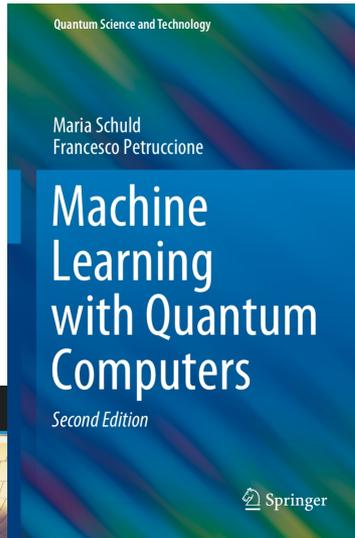
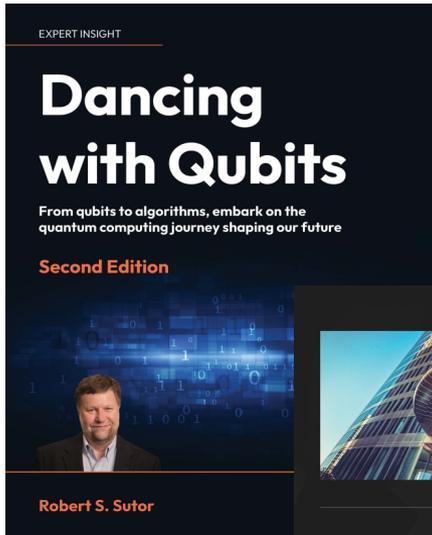
Opportunities that exist today to assist AI in niche areas:

- QC extending “computational scale” as the new HPC for AI
- QC providing high-dim Hilbert state space, e.g. Quantum Kernels > Kernels (Decisions)
- QC facilitating “superposition” search space, e.g. Quantum RL > RL methods (Games)
- QC acting as efficient “stochastic” engine, e.g. Quantum Walks > Random Walks (Finance)
- QC enabling data embedding in Hilbert space, e.g. word embedding for LLMs (Agentic AI)
- QC offering quantum spectral clustering, e.g. unsupervised pattern matching (Sensing / Vision)

Should we be optimistic?

- Yes

Recommended reading on QML with Qiskit



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

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

M. Emre Sahin¹, Edoardo Altamura², Oscar Wallis³, Stephen P. Wood⁴, Anton Dekussar⁵, Declan A. Miller⁶, Takashi Inamichi⁷, Atsushi Matsuo⁸,^{1,2} and Code contributors

¹The Hartree Centre, STFC, Sci-Tech Daresbury, Warrington, WA4 1AD, United Kingdom
²IBM Quantum, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA
³IBM Quantum, IBM Research Europe - Dublin, Ireland
⁴IBM Research - UK
⁵IBM Quantum, IBM Research - Tokyo, Tokyo 105-8510, 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

The convergence of quantum computing and machine learning promises a prospective shift in both research and industry. Quantum machine learning (QML) leverages the principles of quantum mechanics to potentially enhance or accelerate classical machine learning algorithms, opening new frontiers in fields ranging from materials science to finance. As the field of QML matures, there is a growing need for accessible and powerful software tools that bridge the gap between theoretical QML algorithms and their practical implementation on emerging quantum hardware and simulators.

Qiskit Machine Learning (ML)¹, an open-source module within the Qiskit framework [1], addresses this need by providing a comprehensive and user-friendly platform for exploring the exciting landscape of QML. Built on core Qiskit elements such as primitives, it combines quantum circuit design, simulation, and execution to deliver cutting-edge QML algorithms. Users can experiment with quantum enhancements to established methods, such as quantum kernels for Support Vector Machines, or explore new, fully quantum approaches. Its tight integration with Python and reliance on widely used libraries like NumPy [2] and scikit-learn [3] make it accessible to practitioners in diverse fields, from engineering to the life sciences. It also includes a dedicated API connector to PyTorch [4] for neural network-based algorithms, seamlessly bridging quantum circuits with modern deep learning frameworks.

Qiskit ML is freely distributed under the Apache 2.0 license, encouraging community participation and open collaboration. Moreover, it sets itself apart from other platforms like PennyLane [5] in its approach to quantum hardware usage. Specifically, Qiskit ML's architecture is deliberately designed to handle quantum hardware workloads, while also allowing experimentation with

state-of-the-art classical simulators and models of emulated hardware noise from near-term devices. Moreover, it is designed to be modular and extensible, making the addition of new quantum algorithms or building upon existing ones straightforward. Supported by extensive educational resources and tutorials, Qiskit ML stands at the forefront of QML research, helping students, scientists and developers worldwide investigate the applications of quantum computing for machine learning.

This broad suite of quantum acts as concept simulators, ity, moving to a soft and pre. The big works and Reols for getting in Fig. 1 a

ML predominantly depends on Qiskit's primitives. It also interfaces with classical machine learning frameworks such as scikit-learn and Python numerical-core libraries like NumPy, enabling a continuous integration of classical and quantum machine learning techniques. Additionally, the models follow SciPy's structural foundation, and there is functionality for integrating neural networks with PyTorch to support the design, training, and inference of hybrid quantum-classical models.

* stefano.mesa@fc.ac.uk
† github.com/qiskit-community/qiskit-machine-learning

ph] 19 Jun 2024

Quantum computing with Qiskit

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 reprogrammability 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

Quantum computing is progressing at a rapid pace, and robust software tools such as Qiskit are becoming increasingly important as a means of facilitating research, education, and to run computationally interesting problems on quantum computers. For example, Qiskit was of utility gating [51], multi-tolerant identifying a set of Pauli rotations, sparse matrices, empirical of ing gates [101, 104], or other high-level mathematical operators. Importantly, these abstract circuits are representable in Qiskit, which contains synthesis methods to generate concrete circuits from them. Such concrete circuits are formed using a standard library of gates, representable using intermediate quantum languages such as OpenQASM [31].

The pack- a current have com- by of whom the Python well as hant- 8, 4). More have used d Qiskit in is the most [10]. milestone. ign philoso- gn drive into ch firm its capabilities lanthanum a quantum leveraged ing a vari-

II. DESIGN PHILOSOPHY

We begin by discussing Qiskit's scope within the broader quantum computing software stack, as illustrated in Figure 1. Starting from a computational problem, a quantum algorithm specifies how the problem may be solved with quantum circuits. This step involves translating the classical problem to the quantum domain, for example Fermion to qubit mapping [34, 62]. Circuits at this level can be quite abstract, for example only specifying a set of Pauli rotations, sparse matrices, or other high-level mathematical operators. Importantly, these abstract circuits are representable in Qiskit, which contains synthesis methods to generate concrete circuits from them. Such concrete circuits are formed using a standard library of gates, representable using intermediate quantum languages such as OpenQASM [31].

The transpiler rewrites circuits in multiple rounds of passes, in order to optimize and translate it to the target instruction set architecture (ISA). The word "transpiler" is used within Qiskit to emphasize its nature as a circuit-to-circuit rewriting tool, distinct from a full compilation down to controlling binaries which is necessary for executing circuits. But the transpiler can also be thought of as an optimizing compiler for quantum programs.

The ISA is the key abstraction layer separating the hardware from the software, and depends heavily on the quantum computer architecture beneath. For example for a physical quantum computer based on superconducting qubits, this can include CNOT, \sqrt{X} and $BZ(9)$ rotations. For a logical quantum computer, it can include joint Pauli measurements, magic state distillation, or other operations specific to the error correction code [25]. Note that the ISA is often more than just a universal set of quantum gates, and can include measure, reset or de- lay operations, or classical control-flow such as if/else

Qiskit 2.3.0

qiskit-bot released this Jan 9 · 99 commits to main since this release

Dec 24, 2025
edotamura
0.9.0
1470e fa

Qiskit Machine Learning 0.9.0

This release is primarily a compatibility and migration release, bringing Qiskit Machine Learning forward to the Qiskit 2.0 / V2 primitives ecosystem, while also delivering API enhancements (notably in classifiers and optimizers), tightening supported Python versions, and reducing the optional dependency surface.

Summary and thank you!

- QML is an intersection of QC x ML x Maths
- The most common approach to PQC training are VQAs
- Quantum encoding is the key to success (but full of traps)
- Measurement of circuits requires interpretation of results
- Quantum circuit design needs to consider what happens in Hilbert space and what the optimizer does in classical parameter space, both are in conflict
- Training of the hybrid quantum-classical circuit relies on a classical optimizer, and its execution on a quantum machine
- Backpropagation does not work on quantum machines, due to: measurement collapse and no-cloning theorem
- Quantum models are highly sensitive to initialisation, so their performance needs to be assessed across different model instances
- Dimensionality of Hilbert space and parameter space promotes the circuit expressivity, yet, hampers the circuit trainability
- Qiskit QML models utilize PQCs
- Qiskit provides tools for data encoding, ansatz design and measurement
- Qiskit provides powerful runtime framework for training sampling (classification) and estimation models, equipped with noise suppression and mitigation tools
- Adapting ML methods to QML has not yet shown an advantage
- So far, there are no credible demonstrations of QAI > QML

Q&A

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



This presentation has been released under the Creative Commons CC BY-NC-ND license, i.e.

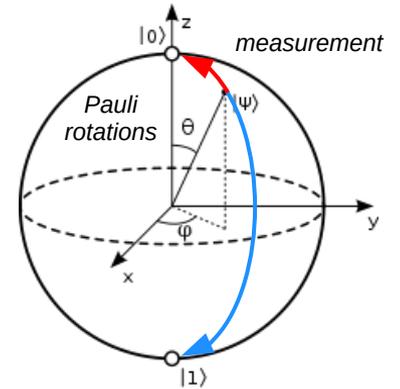
BY: credit must be given to the creator.

NC: Only noncommercial uses of the work are permitted.

ND: No derivatives or adaptations of the work are permitted.

Appendix

With some interesting extras



Who is doing what and where in QML / QAI?

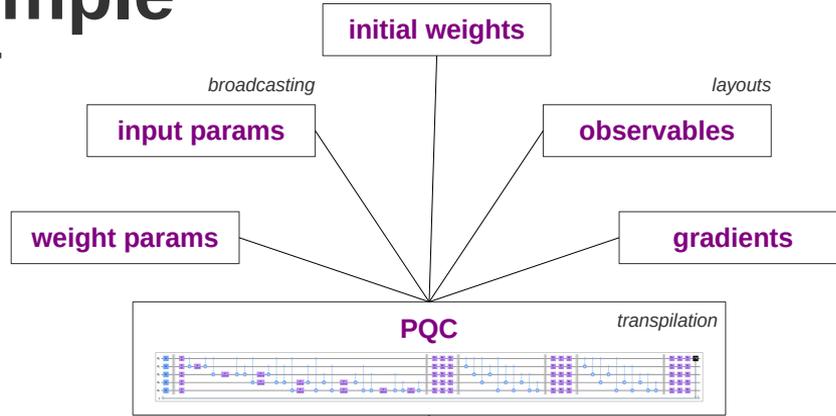
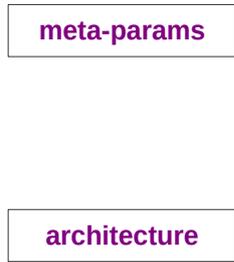
Category Name	Company Name	Sample Project	Continent or Region
Business & Logistics	D-Wave Quantum	Real-time supply chain and fleet routing	North America
	Quanmatic	Semiconductor manufacturing workflow optimization	APAC
	Zapata Quantum	Enterprise generative AI and industrial optimization	North America
	Fujitsu	Digital annealing for traffic and logistics AI	APAC
	QuantumSouth	Cargo load and payload distribution optimization	South America
Finance	Multiverse Computing	Risk management, credit scoring, and stress testing	Europe
	IBM Quantum	Quantum fraud detection and options pricing	North America
	Scenario X	Real-time financial stress testing and risk modeling	Europe
	Horizon Quantum	Automated compilation of classical code into QAI	APAC
	Q-Africa	Financial inclusion and credit-scoring for unbanked	Africa
Chem & Pharma	Microsoft	AI agents for molecular screening and discovery	North America
	Qubit Pharmaceuticals	Physics-driven drug discovery foundation models	Europe
	Algorithmiq	Quantum noise mitigation for drug-target binding	Europe
	Quantum Intelligence	Neural-network based analysis of ADME/Tox	APAC
	G42	Genomic research and Arabic Large Language Models	Arab World
Engineering	SECQAI	Computational Fluid Dynamics (CFD) for aerospace	Europe
	GenMat	Generative materials science for high-perf alloys	North America
	BQP	Quantum-inspired engineering and CAD simulations	India
	Altransinnov	Predictive monitoring for energy infrastructure	Europe
	Senfio	Precision agriculture and yield prediction	South America
Quantum Brilliance	Edge Quantum AI for robotics and satellites	Australia	

Who is doing what and where in QML / QAI?

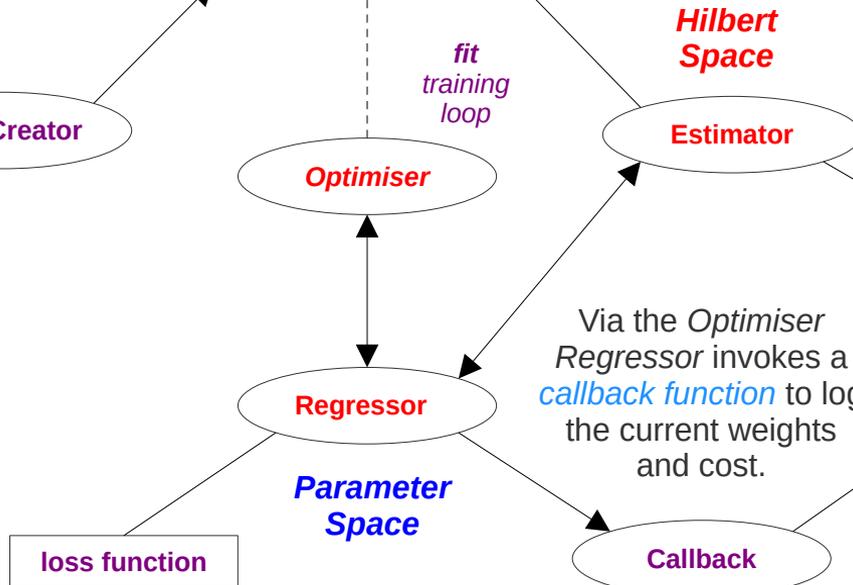
Category Name	Company Name	Sample Project	Continent or Region
Defense & Security	SandboxAQ	GPS-denied navigation (MagNav) and PQC	North America
	Infleqtion	Edge QAI for RF receivers and signal intelligence	North America
	ID Quantique	Quantum Key Distribution and secure backbones	Europe
	QuSecure	Post-Quantum Cryptography (PQC) orchestration	North America
	TII	Sovereign quantum cryptography and security	Arab World
Science & Research	Q-CTRL	Quantum-enhanced navigation and mission planning	Australia
	Xanadu	Differentiable programming and QNN frameworks	North America
	Pasqal	Graph Neural Networks (GNN) for climate and physics	Europe
	NVIDIA	GPU-accelerated QML simulation for field theory	North America
	QuantX Labs	High-precision timing and quantum sensor	Australia
	Google Quantum AI	Quantum Reinforcement Learning and Field Echoes	North America
Sovereign AI Hubs	SAQuTI	Environmental monitoring and biodiversity analysis	Africa
	QpiAI	QAI platform for life sciences and finance	India
	TCS	Hybrid QAI algorithms for global retail chains	India
	Terra Quantum	Hybrid QML for enterprise-grade applications	Europe
	NEC Corporation	5G/6G network traffic optimization	APAC
	QuantumNexis	QAI-powered healthcare analytics platforms	Arab World
	CSIRO	Health, space, mining and defense science	Australia

Training a simple TS Qiskit estimator

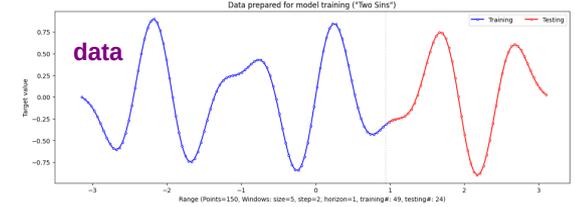
Qiskit **Optimiser** provides function **fit** which executes a training loop, performing: a **forward** pass which applies the model with its current parameters to training data, **loss function**, and a **backward** pass to improve the model parameters.



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.



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



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.

```

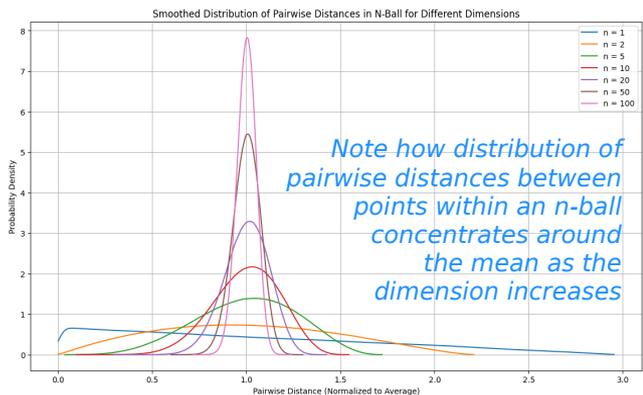
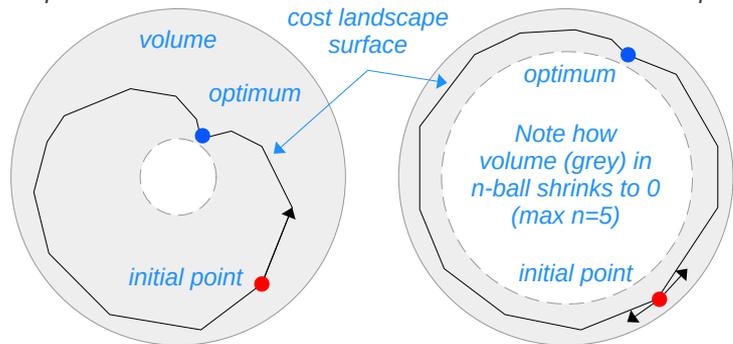
Model training started
training log
(00:00:00) - Iter#: 0 / 500, Cost: 0.238564
(00:00:07) - Iter#: 50 / 500, Cost: 0.162685
(00:00:14) - Iter#: 100 / 500, Cost: 0.126066
(00:00:21) - Iter#: 150 / 500, Cost: 0.073866
(00:00:29) - Iter#: 200 / 500, Cost: 0.053152
(00:00:36) - Iter#: 250 / 500, Cost: 0.038513
(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
    
```

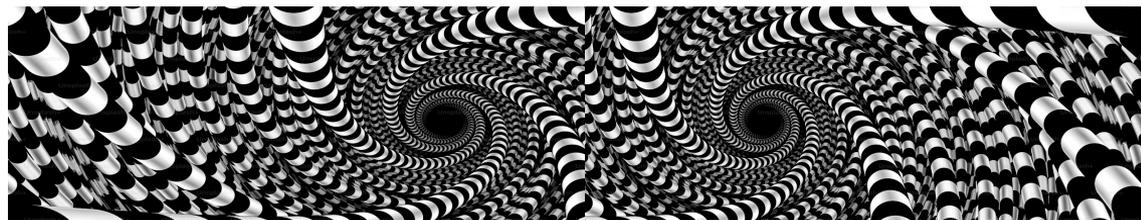
The curse of dimensionality

4-D space

45-D space



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 values are 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)