Objectives for this session:

To briefly explore and explain Quantum Data Science and Quantum Machine Learning all in less than 30 mins

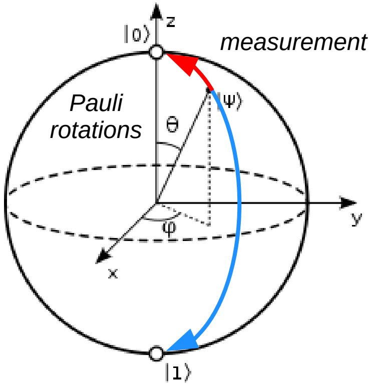
QDS vs. QML vs. QAI and their aims
ML vs QML and QML applications
Parameterised Quantum Circuits (PQC)
Variational Quantum Algorithms (VQA)

QML principles via Quantum AE

- *Model function and structure (symmetry, info loss and info preservation)*
- *Model training / optimisation (reversibility, gradients and nondeterminism)*

QML readings
Summary

Introduction to Quantum Data Science / QML



Jacob L. Cybulski
Enquanted, Australia
School of IT, Deakin University

Quantum Data Science vs. Quantum Machine Learning

Quantum machine learning: ***tools***

is a field of study applying methods and theories of quantum mechanics,

to redefine machine learning concepts and techniques,

for developing and applying new ***learning tools, algorithms and models***

Quantum Data Science: ***knowledge***

is a field using ***scientific methods*** + principles of quantum mechanics,

to create new approaches to extracting and analysing insights, patterns, and meaning from data,

for problem solving / decision-making

Quantum AI: ***application***

is a field focusing on applying quantum computing,

to develop new highly effective technologies,

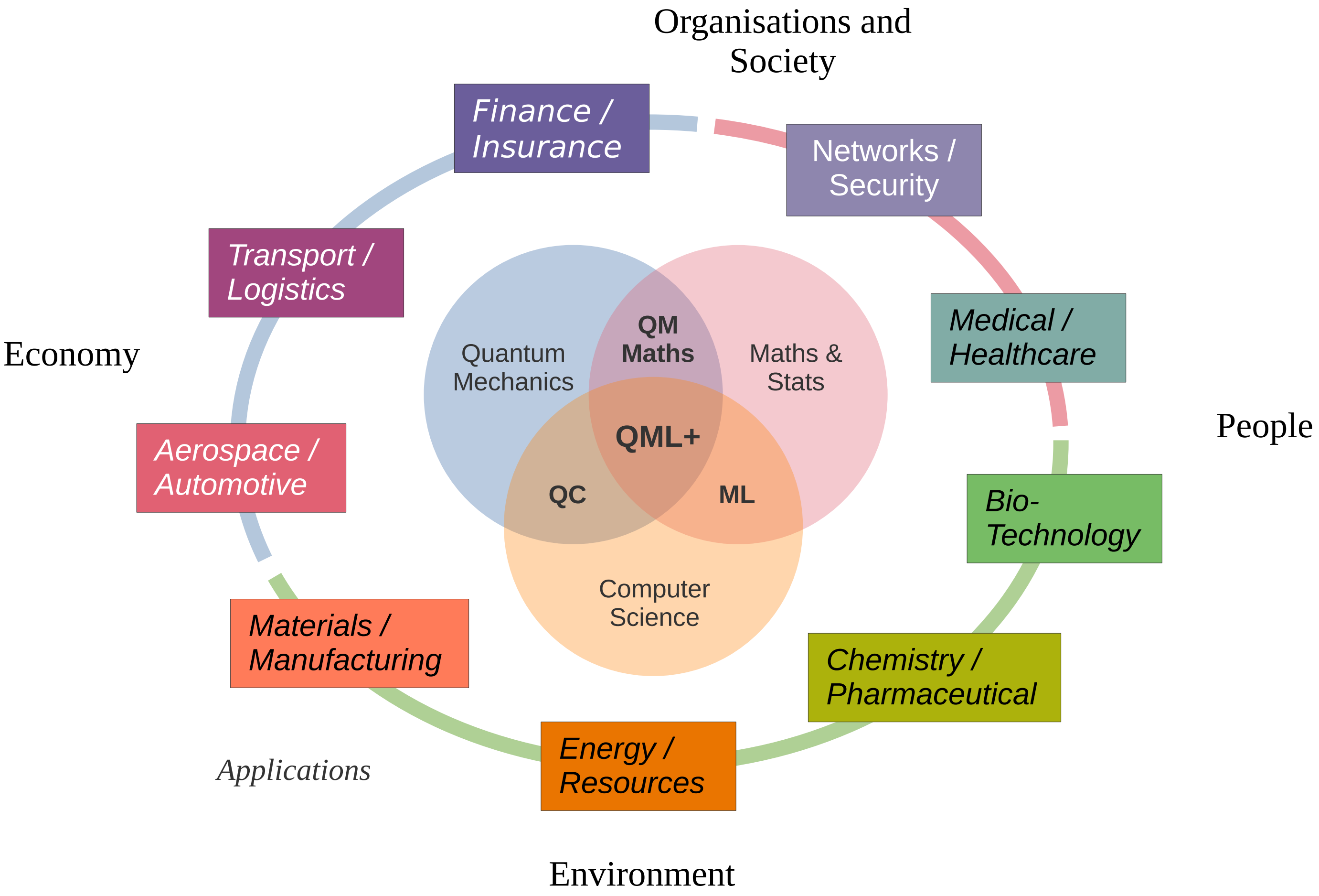
for performing tasks attributed to human cognition, with ***effectiveness approaching human intelligence***

QML+

Quantum ML+

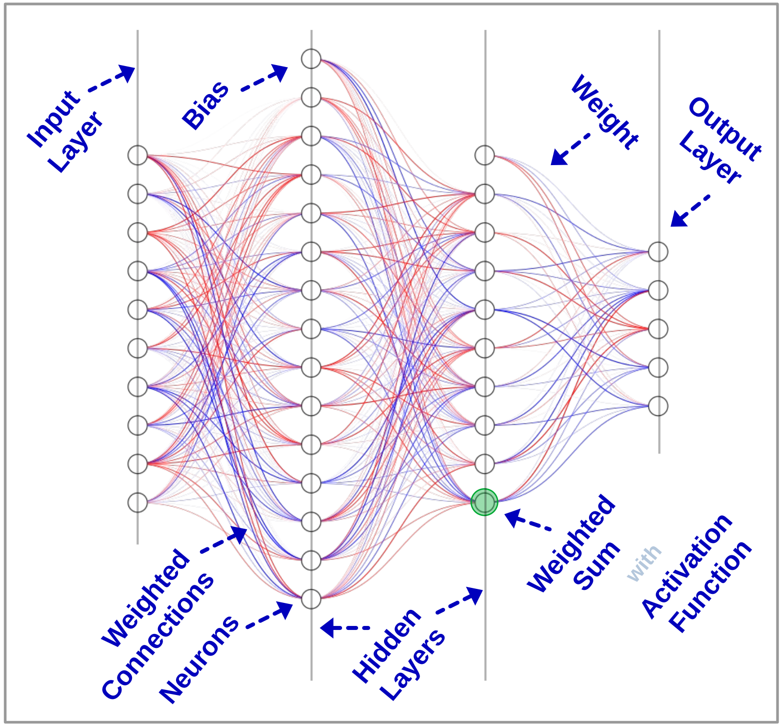
aims of this session

In this session we will explain the principles of Q ML / Q Data Science

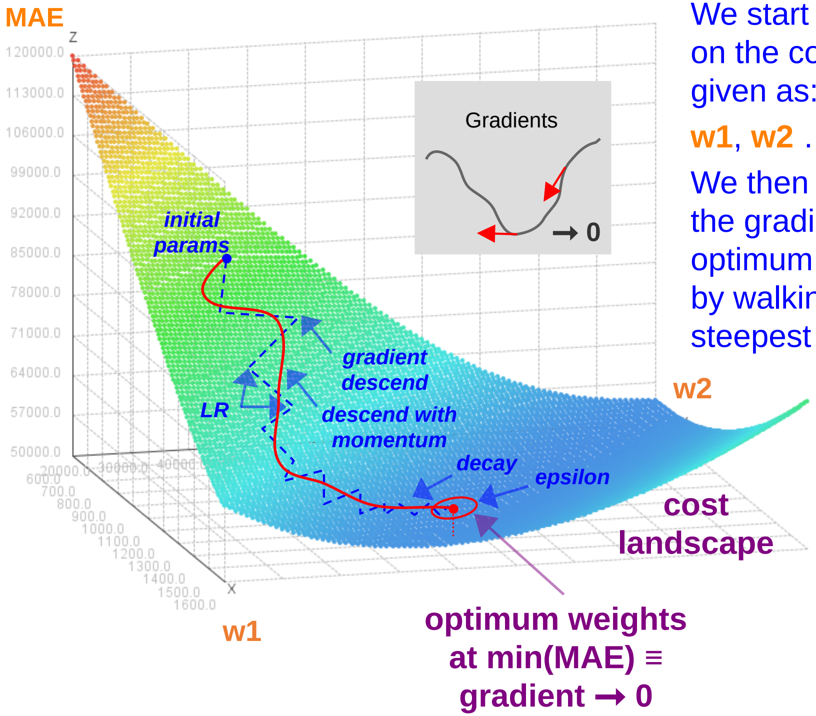


Multilayer Perceptron (MLP)

A class of Neural Network (NN) models



Now we will look for the best NN model, i.e. the model weights which generate the model of the lowest cost (e.g. MAE)



Gradient descent:

We start at any point on the cost landscape given as:

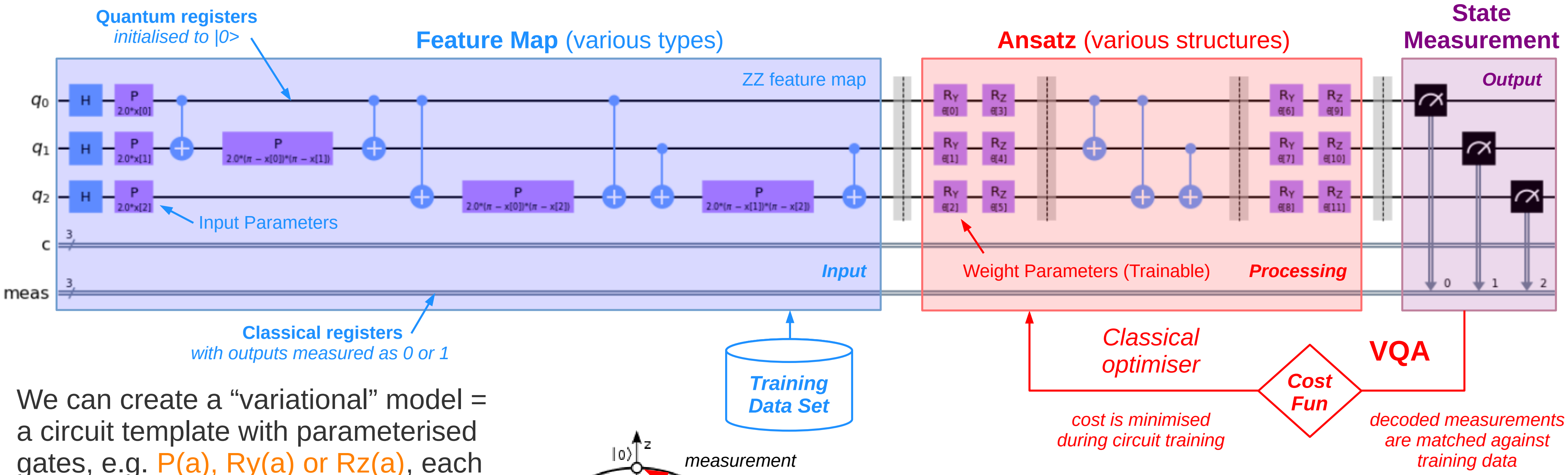
w1, w2 … and MAE

We then search for the gradient=0, i.e. optimum weights, by walking down the steepest slope

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

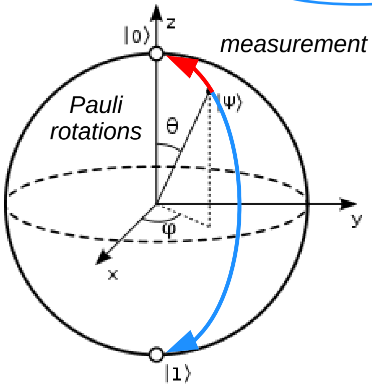
*The majority of circuit operations are **unitary**, i.e. they are reversible and norm-preserving.*

Measurement and qubit reset are not unitary.



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).

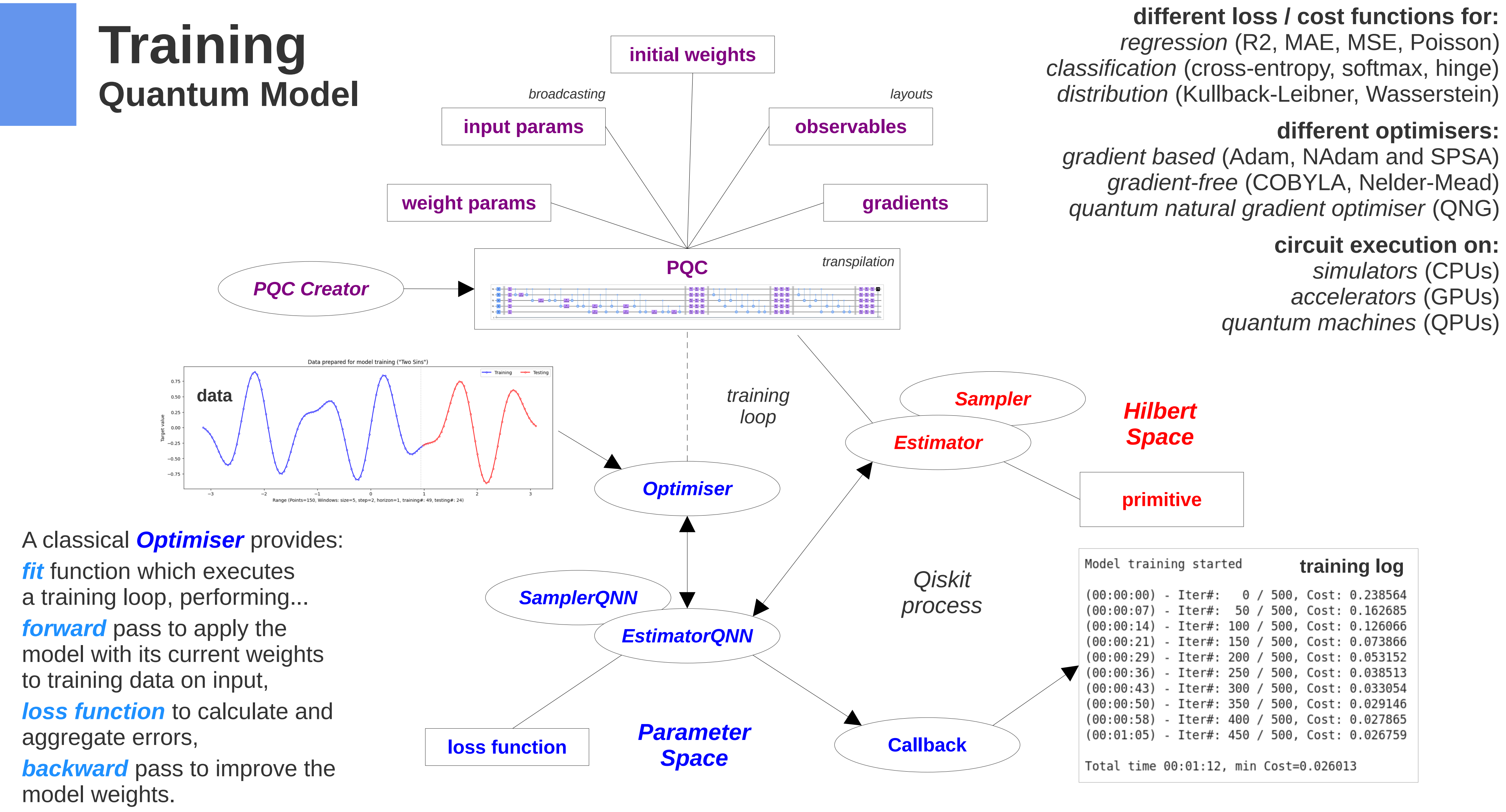
Typically (but now always), circuits consist of three functional blocks, i.e. feature map, ansatz and measurements (not necessarily in this order).



Feature map encodes (embeds) classical input data into its input parameters, setting the model’s initial quantum state.

Ansatz evolves the quantum state, it consists of trainable quantum gates, weight params trained by a classic optimiser

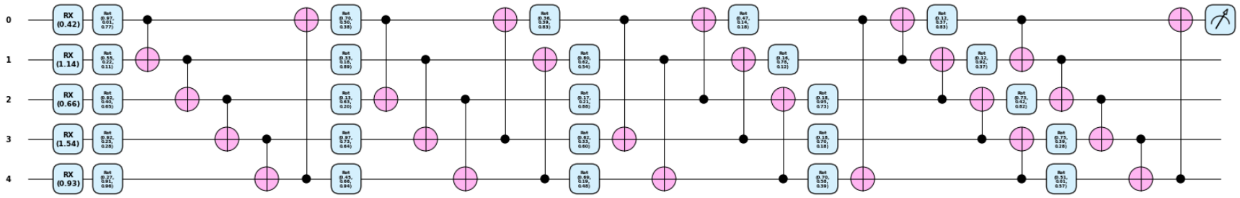
Measurements determine the circuit final state and convert it into the model’s output in the form of classical data.



Training
Quantum Model
initial weights
broadcasting
layouts
input params
observables
weight params
gradients
PQC Creator
PQC
transpilation
Data prepared for model training ("Two Sins")
Training
Testing
data
Target value
Range (Points=150, Windows: size=5, step=2, horizon=1, training#: 49, testing#: 24)
training
loop
Sampler
Estimator
Hilbert
Space
primitive
Optimiser
Qiskit
process
SamplerQNN
EstimatorQNN
loss function
Parameter
Space
Callback
different loss / cost functions for:
regression (R2, MAE, MSE, Poisson)
classification (cross-entropy, softmax, hinge)
distribution (Kullback-Leibner, Wasserstein)
different optimisers:
gradient based (Adam, NAdam and SPSA)
gradient-free (COBYLA, Nelder-Mead)
quantum natural gradient optimiser (QNG)
circuit execution on:
simulators (CPUs)
accelerators (GPUs)
quantum machines (QPUs)
A classical Optimiser provides:
fit function which executes
a training loop, performing...
forward pass to apply the
model with its current weights
to training data on input,
loss function to calculate and
aggregate errors,
backward pass to improve the
model weights.
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

Sample Model Training:

Estimate diabetes progression (data from scikit-learn)



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)
```

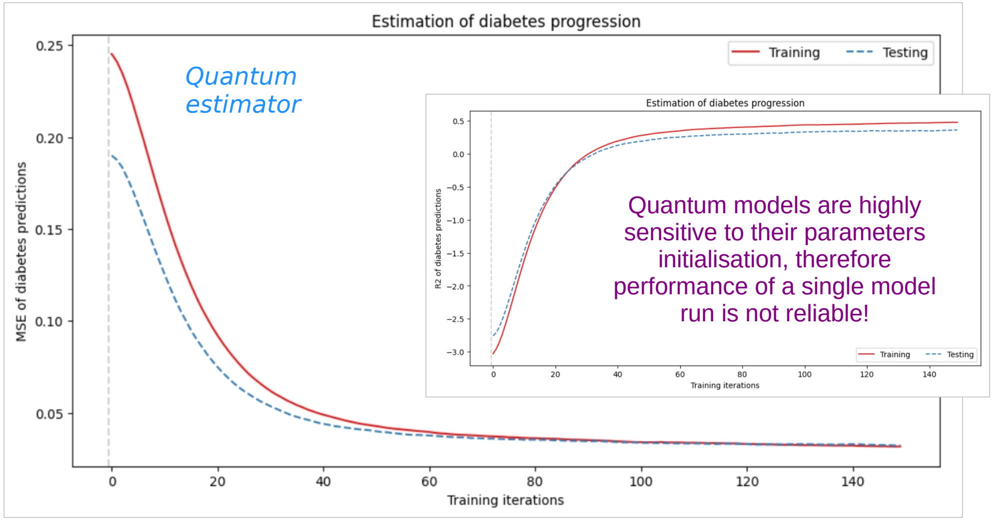
```
device = cpu
samples = 296, features = 5, params = 4721, epochs = 1000
training: cost = 0.0278 @ 0852, r2 = 0.5147 @ 0852
testing:  cost = 0.0304 @ 0980, r2 = 0.4708 @ 0980
elapsed time = 3sec (00:00:03)
```

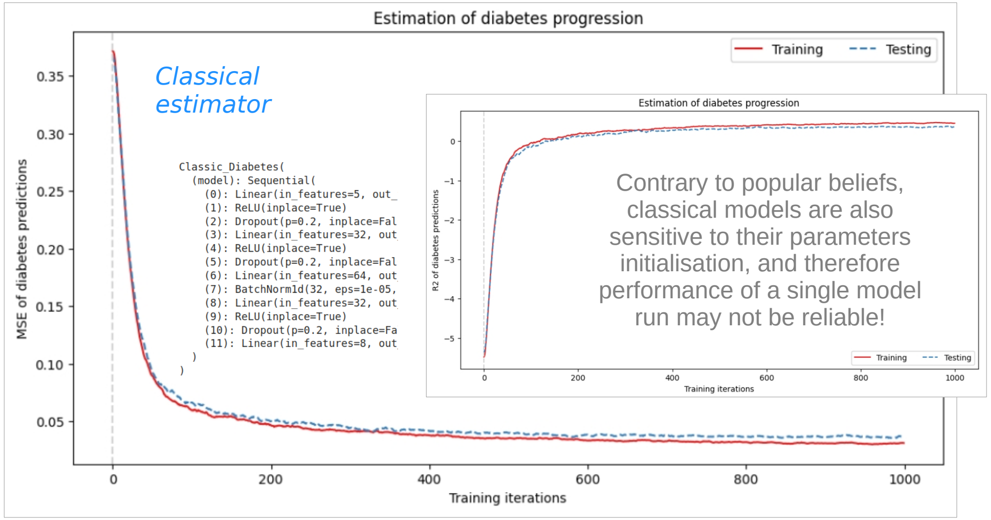
Quantum model training

```
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

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

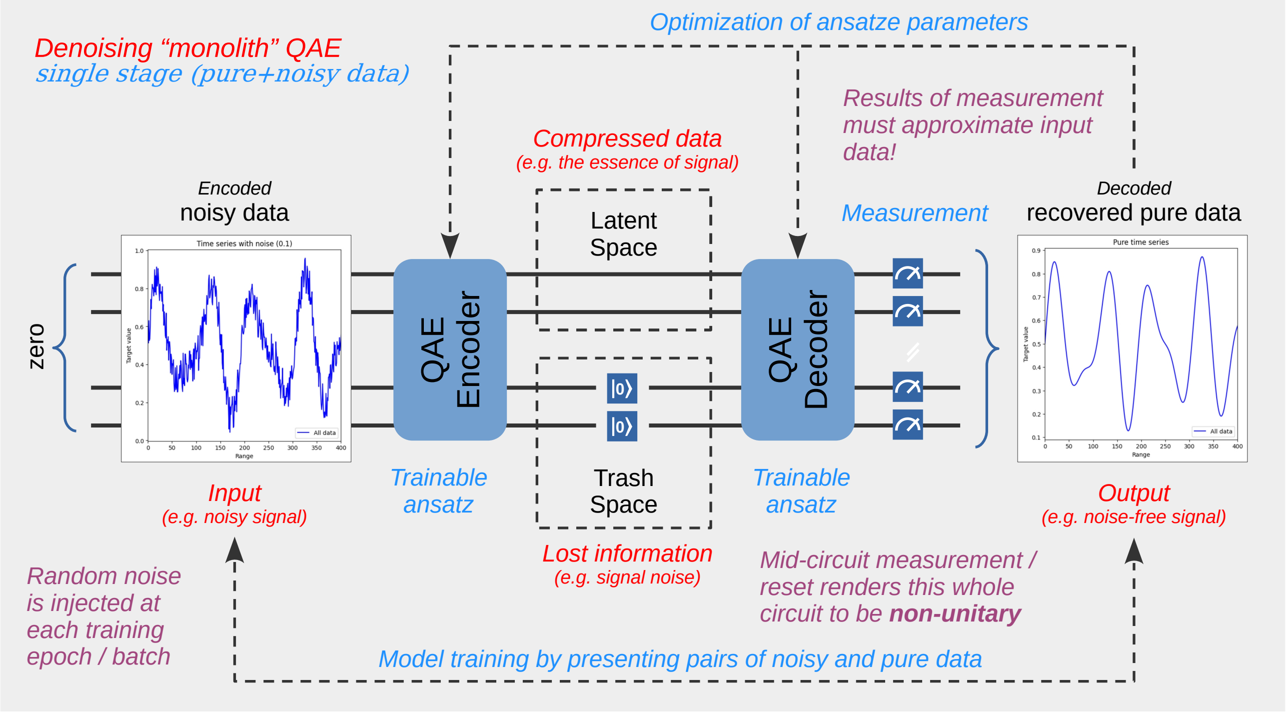




Example: QAE

Denoising Quantum Autoencoder

Definitions and terminology

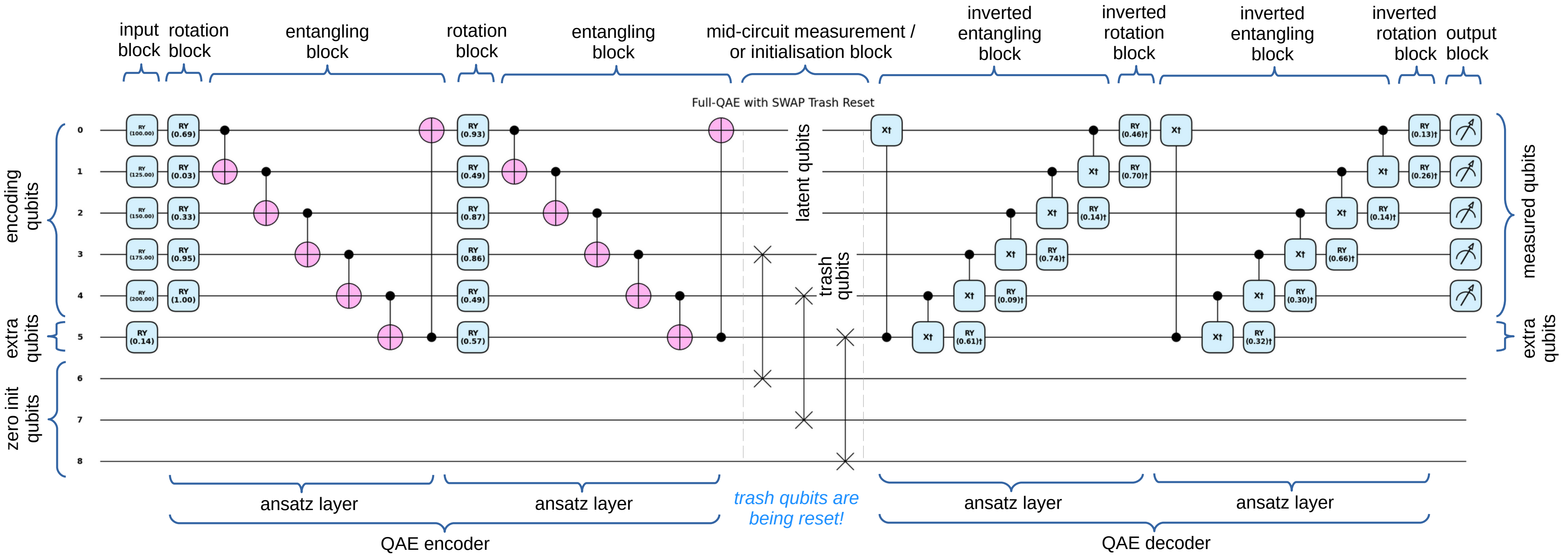


QAEs are quantum models made of the following components:

- *Quantum model:* a circuit of *N* qubits to match time series windows of *N* values
- *Input encoder:* a quantum feature map embedding a window of noisy classical TS
- *QAE encoder:* an ansatz consisting of trainable parameter blocks and entangling blocks, compressing *N* qubits into *n* qubits (n < N)
- *Latent space:* representing the essential features of the window on input
- *Trash space:* representing information lost in QAE training, such as signal noise, reinitialised to prevent flow of information to the decoder
- *QAE decoder:* an ansatz consisting of trainable parameter blocks and entangling blocks, decompressing *n* qubits into *N* qubits (n < N)
- *Output layer:* a measurement block resulting in classical data, which can be interpreted as a TS window of *N* values with reduced noise

Anatomy of the “monolith” QAE

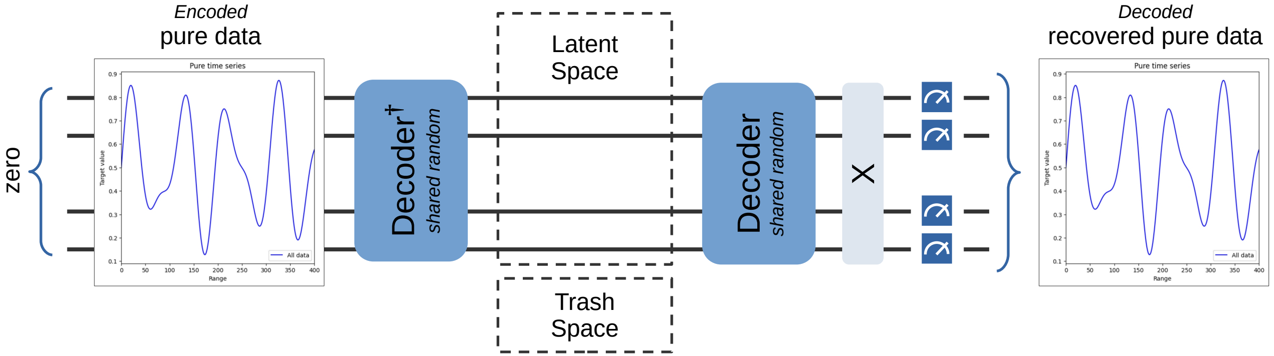
PennyLane circuit

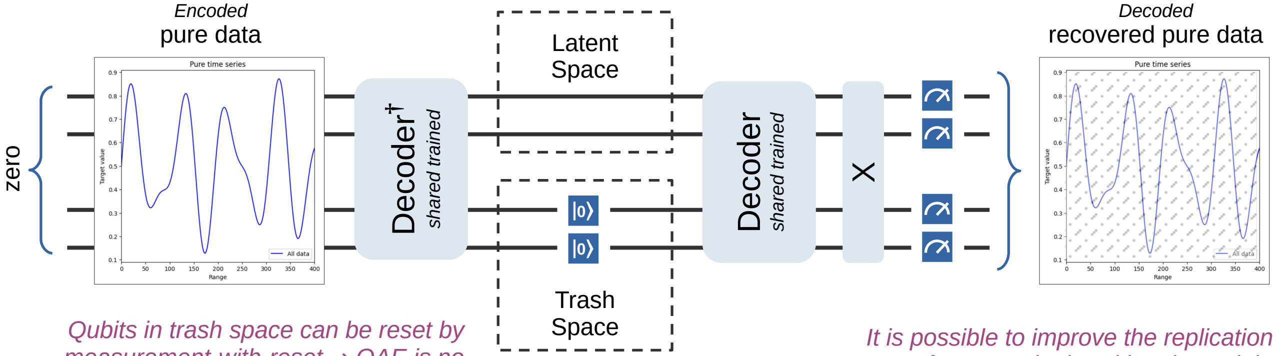


The “monolith” QAE could potentially have lots of weights, so its training can be expensive.
We can reorganise this process by training the QAE encoder and QAE decoder separately.
By doing so, we can gain in training speed, however, likely reducing denoising effectiveness.

Replicating QAE with half-QAE

Consider an angle encoding TS QAE consisting of encoder and decoder unitaries that have mirror image structures, i.e. a QAE Decoder and its adjoint (complex conjugate transpose ≡ inverse) QAE Decoder† used as QAE Encoder.





Qubits in trash space can be reset by measurement-with-reset → QAE is no longer unitary; alternatively with the SWAP operation → QAE stays unitary

It is possible to improve the replication performance by breaking the weight symmetry, however, we are no longer able to rely on the half-QAE training

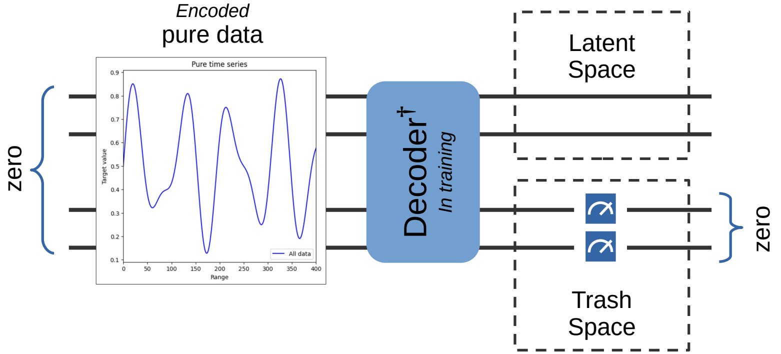
When the latent space spans all qubits, there is no loss of information, and so the QAE is a unitary.

As the QAE Encoder is an inverse of the QAE Decoder, they cancel each other operations.

This means that on output the QAE will always produce an adjoint of its input (upside-down TS).

In angle encoding, this can be corrected with a block of X operations on each qubit.

When we reset trash space, we start losing information flowing from the encoder to decoder.



However, we can train the QAE Encoder / Decoder to reduce this information loss, while preserving their mirror structure and symmetric weights.

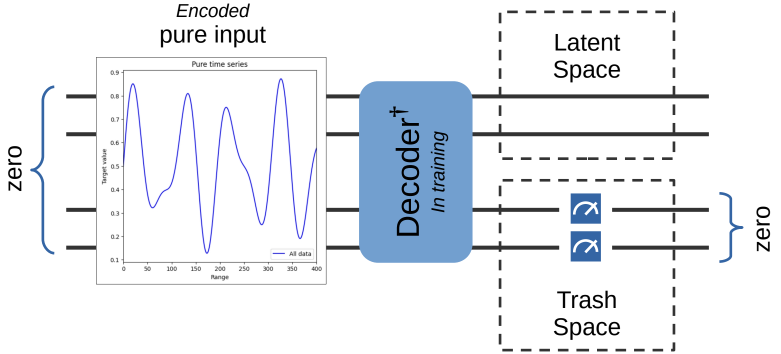
As QAE Encoder is an adjoint of QAE Decoder, **training only QAE Decoder† is enough**, e.g. by ensuring most of its information flows through the latent space, i.e. by converging trash to zero.

Denoising TS with Stacked half-QAEs

There is a widely held belief that replicating QAEs, which can be trained by using only their QAE Decoder, are capable of reducing the signal on input to its most essential information and remove all infrequent, noisy or inessential information.

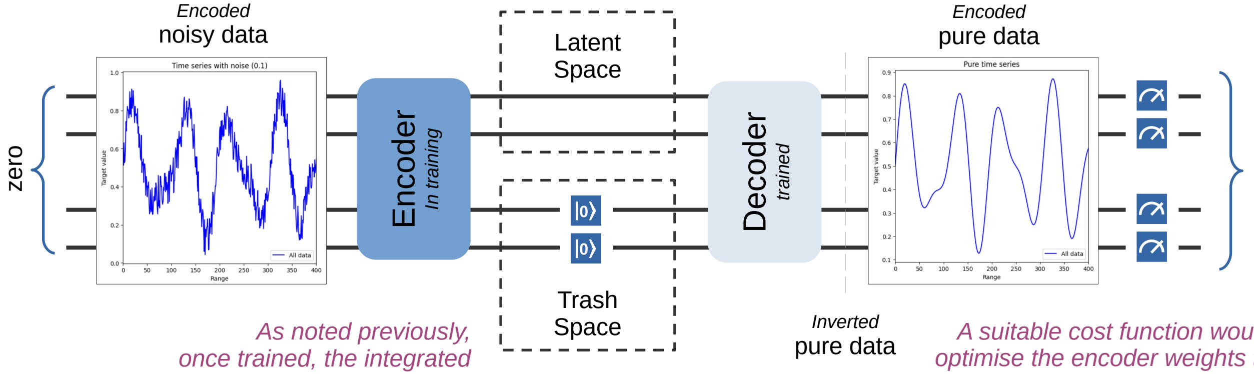
However, replicating QAEs perform poorly in noise reduction.

An alternative is to break weight symmetry and use all QAE weights independently in training.



We will train the denoising QAE in two phases.

Phase 1: Training inverted QAE Decoder using pure data and a cost function aiming to converge trash to zero.



As noted previously, once trained, the integrated circuit will require to invert its output with X on every qubit

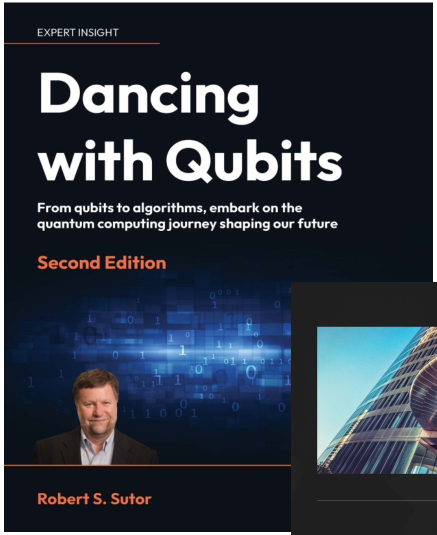
A suitable cost function would optimise the encoder weights to converge output to zero

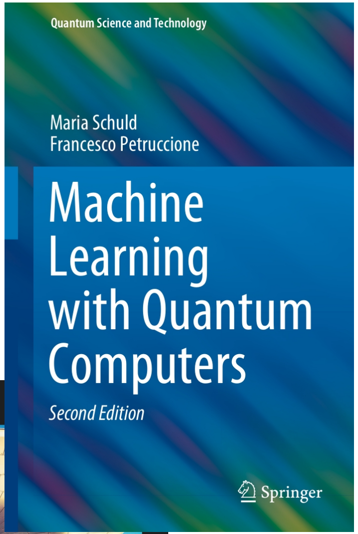
Phase 2: Training a QAE Encoder with noisy data, in combination with an adjoint of the previously trained QAE Decoder, will produce an inverse of pure data approximation.

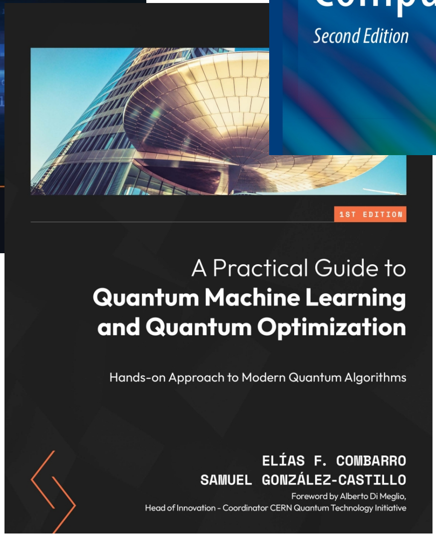
In a perfectly functioning QAE this output would cancel pure data encoding, which can be measured as zero.

Recommended reading

on QML with Qiskit / PennyLane







Summary and Q&A

- QML is an intersection of QC x ML x Maths
- The most common approach to PQC training are VQAs
- 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

- QAE is a quantum model compressing and decompressing information
- QAEs can be used for noise reduction from data on input
- QAEs are great to illustrate quantum model principles
  - Unitaries are reversible and norm preserving quantum operations
  - Standard gates and their combinations are unitaries (except measurement & reset)
  - Application of an adjoint (inverse) $U^\dagger$ of a unitary to U returns identity
  - Information is never lost in unitary circuits (unless measured or decohered)

- In VQA an optimiser is totally blind to what happens in the Hilbert space!

Jacob's recent projects:

Quantum Information Field Theory (QIFT)
combining Quantum Field Theory with Information Processing

Geometry of the Hilbert space
in hybrid quantum-classical models

Quantum time series analysis
processing of quantum time series and signals, e.g. data encoding, measurement, and model training (such as QNNs, QAEs, QRC, etc.)

Quantum model expressivity and trainability
balancing the model's ability to effectively represent data in the Hilbert space, as well as, its capacity to learn and generalize

Impact of barren plateaus countermeasures on QNN capacity to learn
incl. methods of measuring its capacity

Quantum graph theory
exploration of quantum representation and processing of graphs e.g. identifying clusters and communities

Foundations of quantum machine learning
Business value of quantum computing and QML

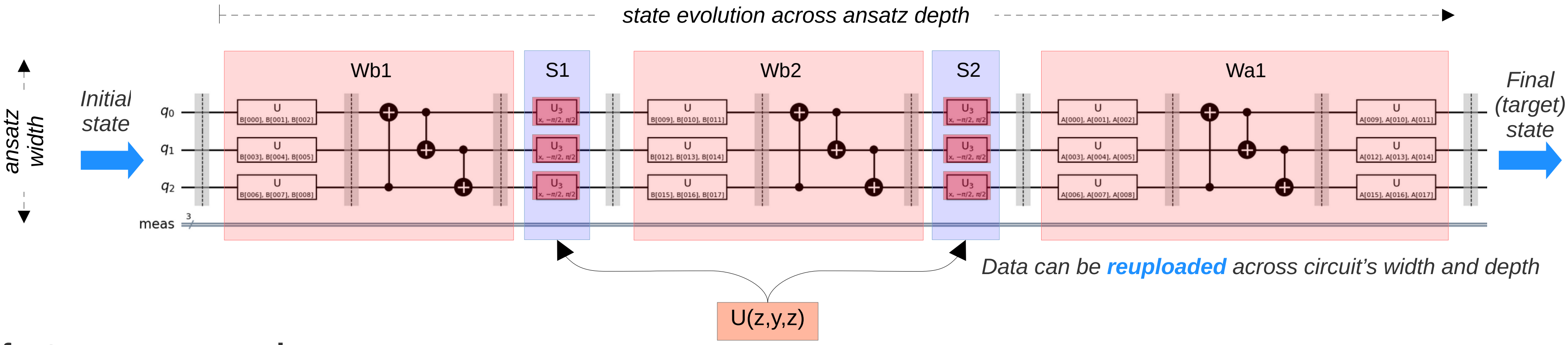
Available resources:

https://jacobcybulski.com/
https://github.com/ironfrown/

Jacob Cybulski,
Founder, Enquanted, Australia
Hon. Assoc. Prof. Deakin, SIT

Appendix

Ansatz design



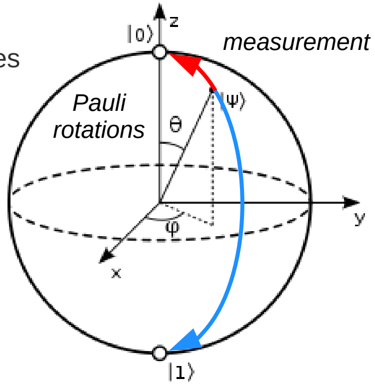
feature maps vary in:
structure and function

ansatze vary in:

- width (number of used qubits)
- depth (longest path from in-to-out)
- dimensions (param #)
- structure (e.g. funnel-in)
- 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 selected weight parameters

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 a measurement doubles the number of outcomes!

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

Geometric Comparison:
Classical vs. Quantum Neural Networks

Feature	**Classical Neural Network (NN)**	**Quantum Neural Network (QNN)**
Fundamental Manifold	Euclidean $\mathbb{R}^n$ (warped by activations)	Complex Projective Space $\mathbb{C}P^{2^n-1}$ (warped by equiv. rel.)
Normalization Strategy	Local/Layer (e.g., Softmax, BatchNorm)	Global (Unitary L_2 norm preservation)
Equivalence Relation	Scale Invariance in ReLU nets $(ReLU(a \cdot x) = a \cdot ReLU(x) \mid a > 0)$	Global Phase Invariance $(\|\psi\rangle \sim e^{i\phi}\|\psi\rangle)$
Metric (Parameter)	Euclidean L_2 Distance	Fubini-Study Metric
Information Geometry	Fisher Information Matrix (FIM)	Quantum Fisher Information Matrix (QFIM)
Loss Landscape	Non-convex, affine-based transformations	Periodic, defined by $SU(2^n)$ rotations
State Representation	Bounded hypercube or open vector space	Generalization of the n-qubit Bloch Sphere