Objectives for this session:

To briefly explore and explain Quantum Machine Learning with PennyLane hands-on demos all in less than 60 mins

What is QML?
Parameterised Quantum Circuits (PQC)
Variational Quantum Algorithms (VQA)
Quantum data encoding
Quantum state measurement
Why using PennyLane?
Training a PennyLane model
Sample model training
- *Why did it succeed and then failed?*
- *Data encoding nightmares*
- *How was the model improved?*

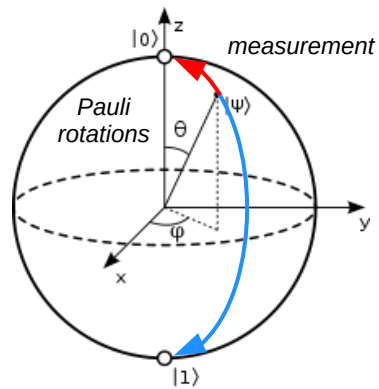
QML readings
Summary

Introduction to QML+ (with hands-on in PennyLane)

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

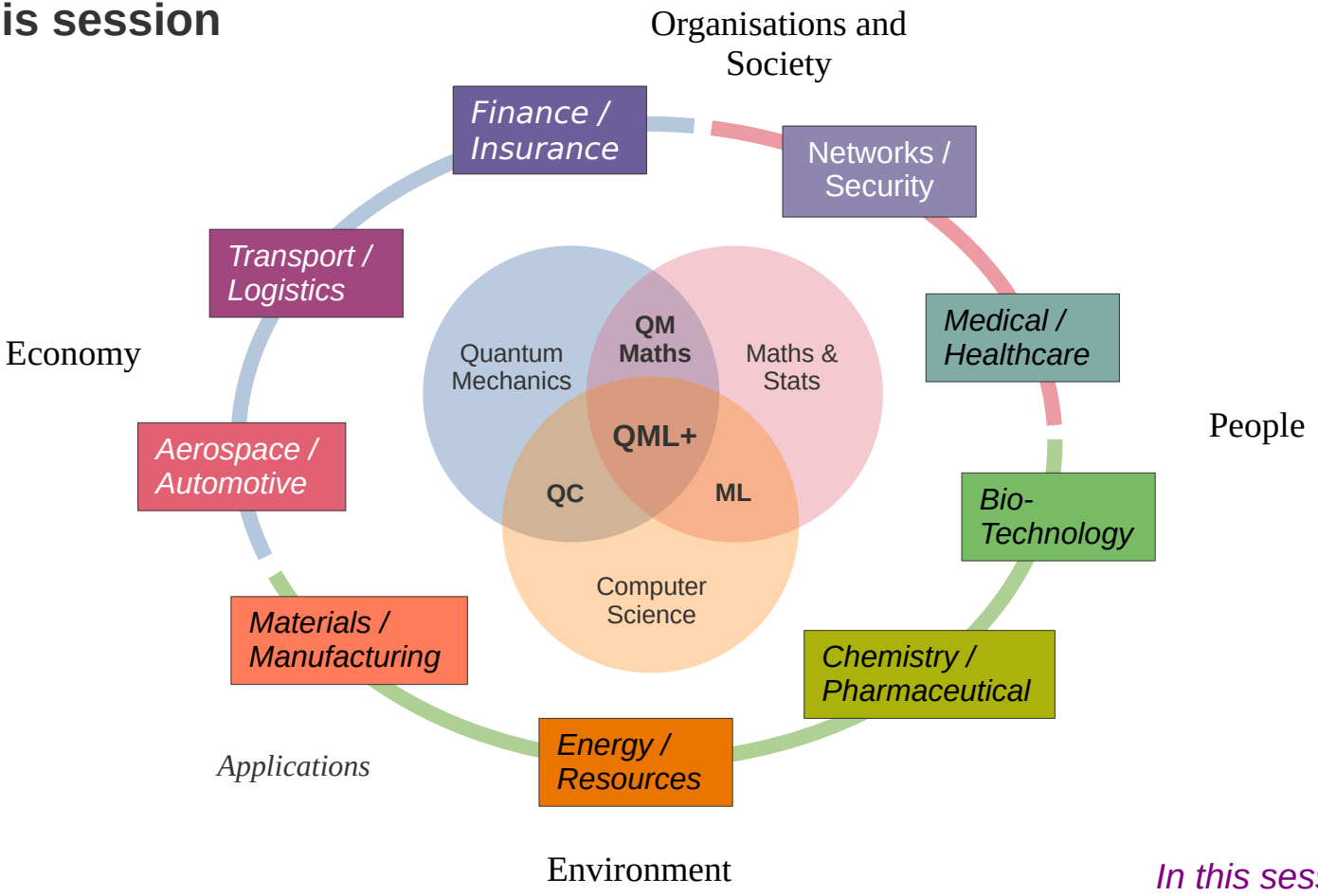
and

Fatima Ansarizadeh
School of IT, Deakin University



Quantum ML+

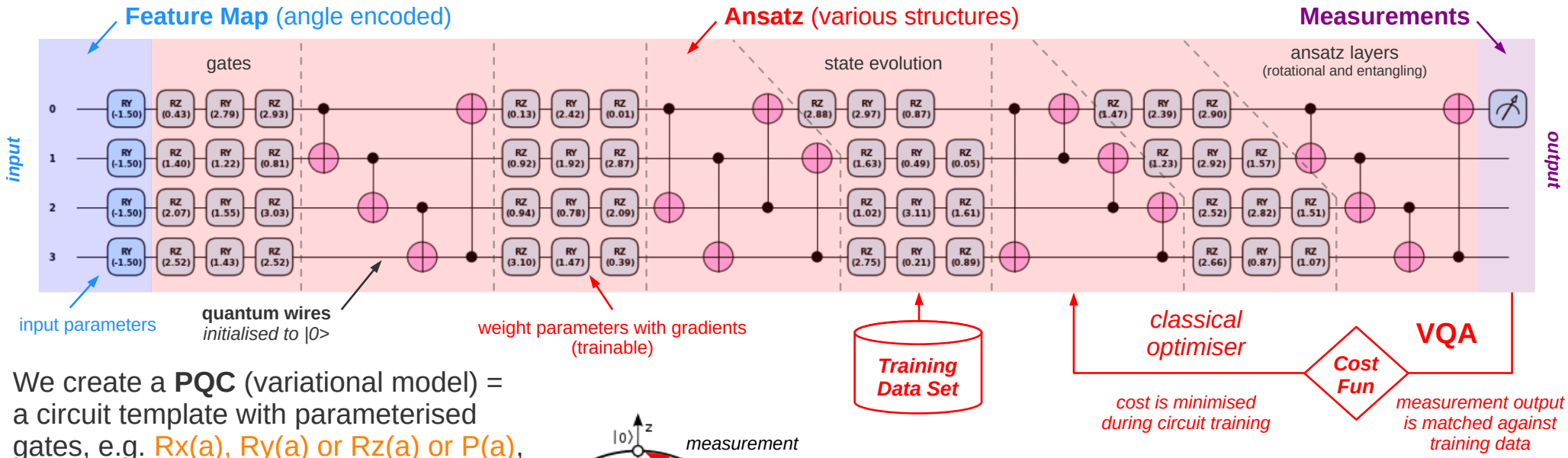
aims of this session



In this session we will explain the practice of QML using PennyLane

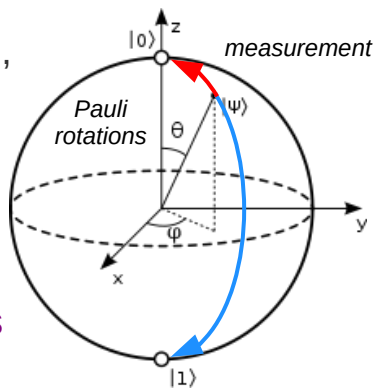
Parameterized Quantum Circuits (PQC) & Variational Quantum Algorithms (VQA)

*In PennyLane a **Circuit** is a function.*
To become executable on a specific device,
*it needs to become a **QNode** (quantum node).*



We create a **PQC** (variational model) = a circuit template with parameterised gates, e.g. Rx(a), Ry(a) or Rz(a) or P(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).



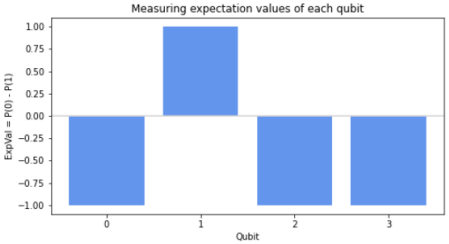
Feature map encodes (embeds) classical input data into its input parameters, setting the model's initial quantum state.

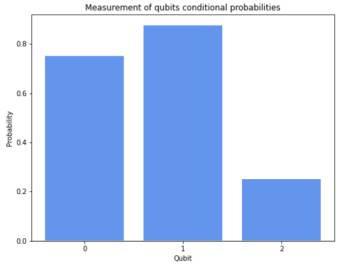
Ansatz evolves the quantum state, it consists of trainable quantum gates, weight params trained by a classic optimiser

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

Data Encoding

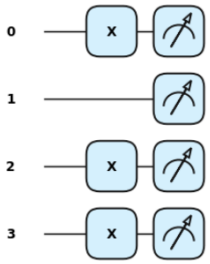
Maria Schuld and Francesco Petruccione
Machine Learning with Quantum Computers. 2nd ed. Springer, 2021.





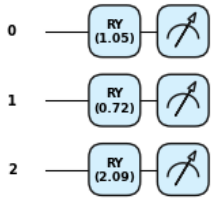
Encoded values:
arccos(0.5)
arccos(0.75)
pi-arccos(0.5)

Basis encoding



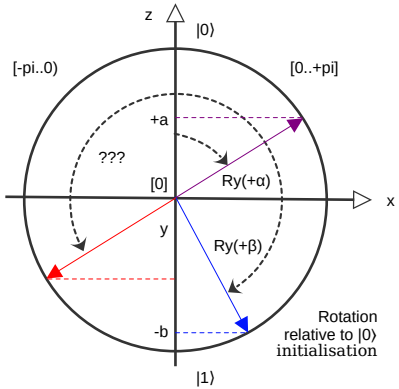
The simplest data encoding and very popular, however, little data can be encoded at a time, you can encode only binary values per each qubit.

Angle encoding



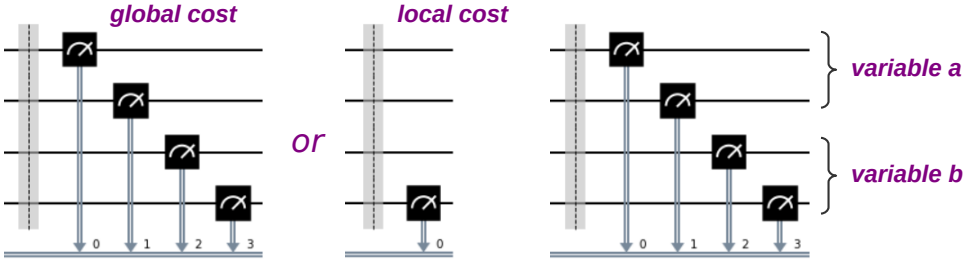
One of the most flexible and very effective, you can encode floating point numbers.

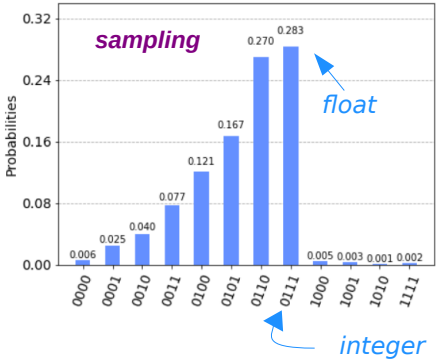
What you encode depends on your intention!



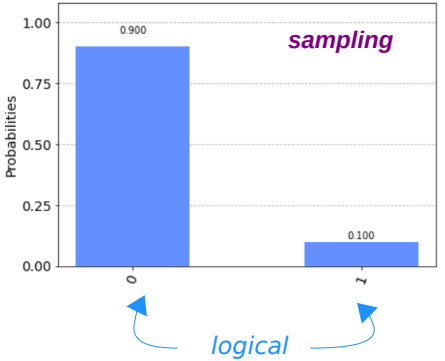
There are many encoding methods, e.g. basis, angle, amplitude, QRAM, …

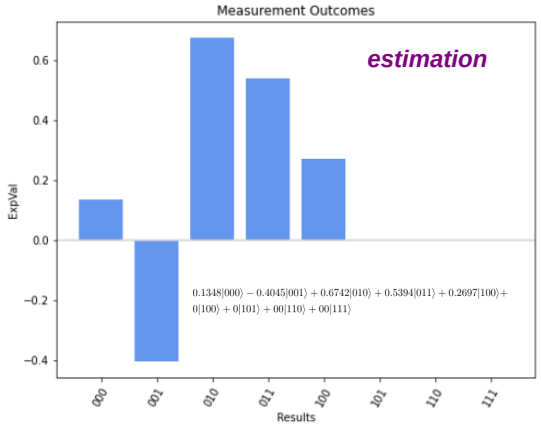
Encoding can be repeated across the circuit, which is called data reuploading





or





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

Why PennyLane?

Everything is a function!

PennyLane (PL) …

- Supports *differentiable programming paradigm*
- Integrates seamlessly with *Python*
- Has a range of operations for *state preparation*, *gates*, *measurements* and *models*
- Supports creation of flexible *quantum algorithms*
- Executes on *simulators* and *quantum hardware*
- Supports *error mitigation*
- Extends its built-in *system of gradients* with those from JAX, PyTorch, Keras, TensorFlow, or NumPy
- Allows training with *hardware-compatible gradients* and *higher-order derivatives*
- Supports *hybrid quantum-classical models*
- Provides just in time *compilation* and *optimisation*
- Can be extended with models and optimisers from other SDKs, e.g. *PyTorch* and *TensorFlow*

PennyLane Demo:

- Create a simple PL model to fit a simple function
- Learn to initialise model weights
- Explore the impact of ansatz structure on performance
- Create minimalistic quantum models
- Learn the interaction of data encoding and ansatz
- Investigate different types of entangling
- Apply the best solution to more complex data
- Learn about stamina and wisdom in QML development

Key takeaways:

- Plan model development, tests and experiments
- Bad data encoding spoils the bunch!
- Strong entanglement improves the data fit
- More width and depth = the curse of dimensionality
- Carefully consider your quantum model initialisation
- Surprise - a single qubit model still works! (and well)
- More training does not solve the problems
- Data reuploading makes a huge difference!

Fundations of PennyLane

Fatima Ansarizadeh

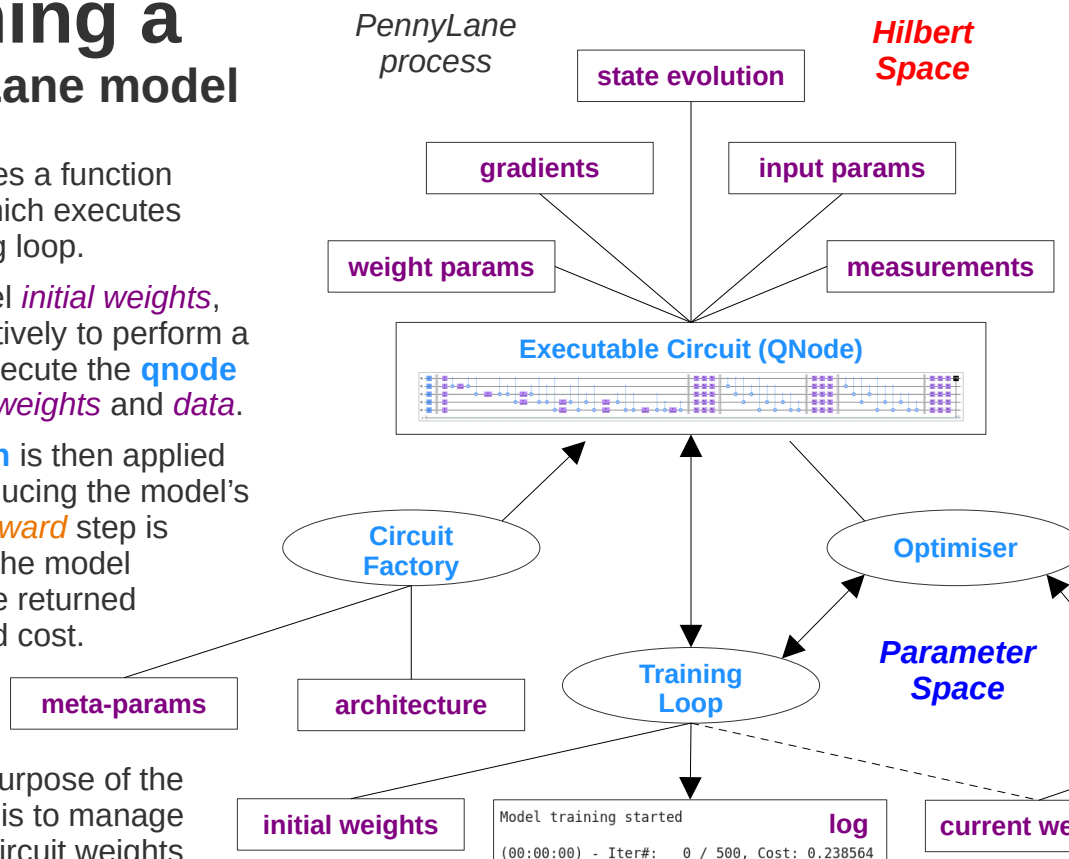
Let’s explore the foundations of PennyLane

Training a PennyLane model

Optimiser provides a function *step_and_cost* which executes a step in a training loop.

It starts with model *initial weights*, and is called iteratively to perform a *forward* step to execute the **qnode** using the *current weights* and *data*.

The **cost function** is then applied to the results producing the model's cost. Then a *backward* step is taken to improve the model weights, which are returned with the calculated cost.



The purpose of the **training loop** is to manage and improve the circuit weights and collect the costs and models weights in the *log*.

The loop starts with the model's *initial weights* and then passes the **cost function**, *current weights* and *input data* to the **optimiser** one step at a time.

A dataset of (X, y) pairs needs to be prepared, cleaned and partitioned for training and testing.

In PennyLane the PQC is typically written as a **circuit factory** - a function defining the circuit *meta-parameters* and generating a **closure**, a function operating in the factory's environment, and responsible to defining the circuit *input parameters*, *weight parameters*, and most importantly its overall *architecture* and logic guiding the *evolution of its state*.

The circuit function is then deployed on a physical device to create an executable circuit, referred to as a **qnode** (quantum node).

When executing, the qnode takes values of weights and inputs (possibly in batches) and returns measured expectation values.

Circuit weights also define *gradients* which are used by the **optimiser** during the optimisation process and to calculate expectation values.

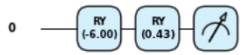
A qnode can be deployed on a variety of devices (e.g. a simulator "default.qubit", GPU "lightning.gpu" or real QPU, e.g. "ionq.qpu"), and optionally interfaced with various gradient packages (e.g. "torch") and differentiation methods (e.g. "adjoint").

Sample model training

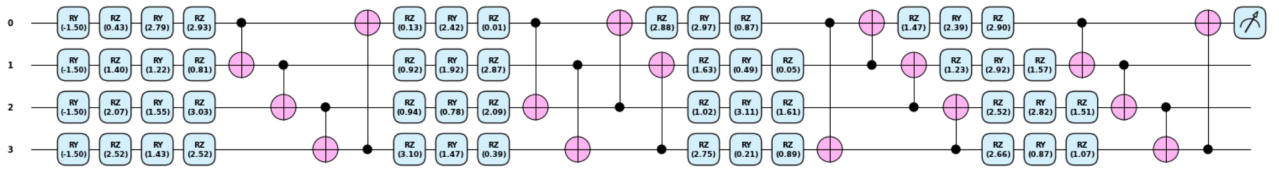
Sweat and tears (with synthetic data)

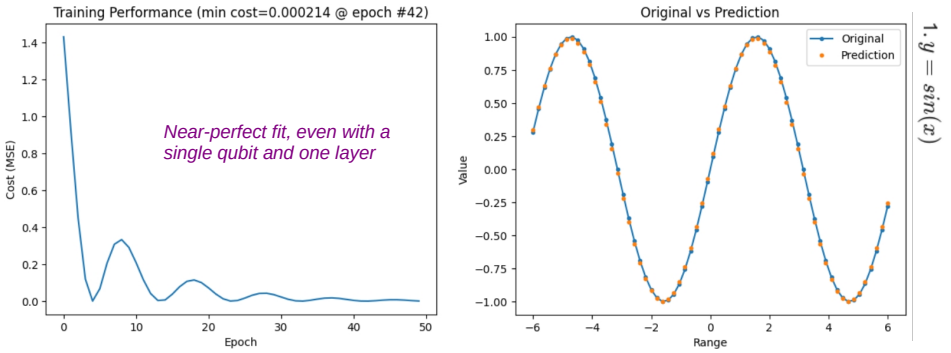
This example can be found at:
https://github.com/ironfrown/pennylane_intro_hands_on/

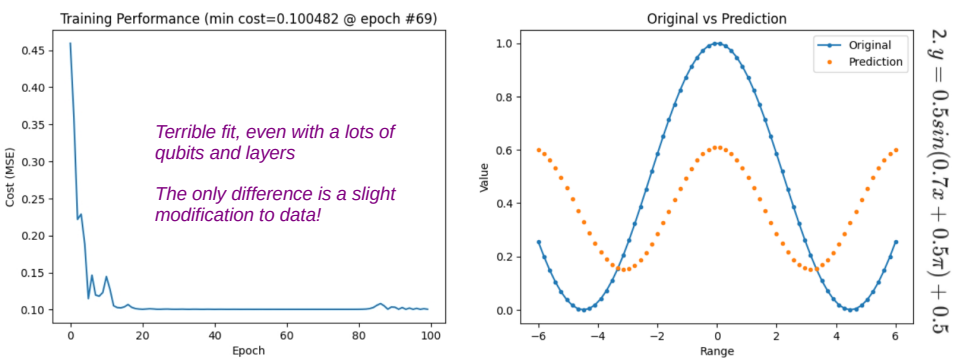
The minimal, yet near-optimal circuit



The best circuit for the first dataset, but not working for the second dataset





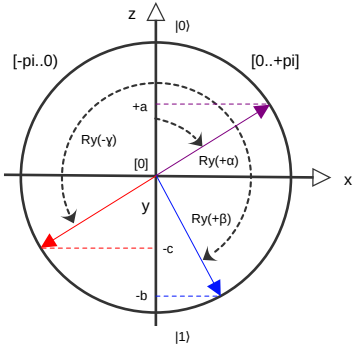



```
Training completed: epochs=50, min cost=0.000214 @ 42, time=002 secs
```

```
Training completed: epochs=100, min cost=0.100482 @ 69, time=028 secs
```

We trained the model for simple data.
It worked perfectly well.
Even with the simplest model!

Why did it work so well?



We applied this model to a very similar dataset.
We experimented with weight initialization,
epoch#, qubit#, layer#, entanglements.

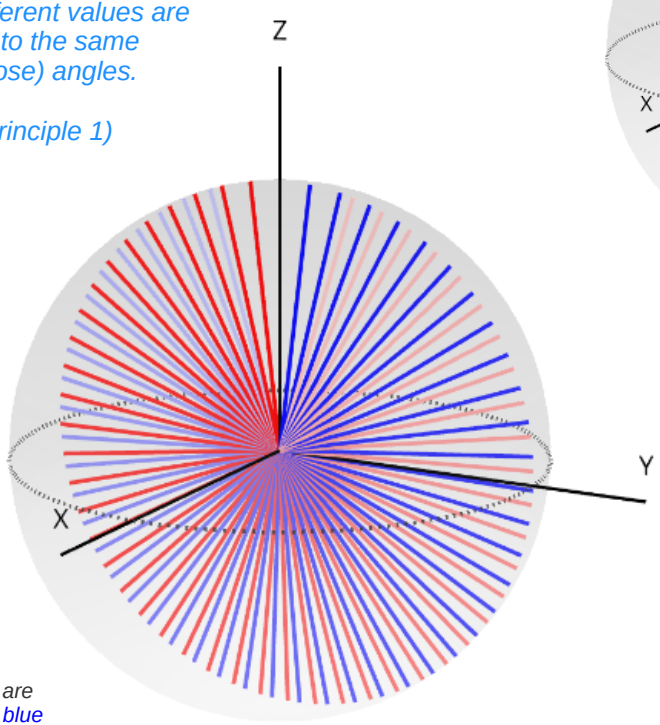
All failed! Why did it fail so badly?

Angle encoding

The Good, the Bad and the Ugly

This example shows encoding values wrapping around the Bloch sphere (e.g. -6 to 6), so that different values are mapped into the same (or very close) angles.

(violates principle 1)

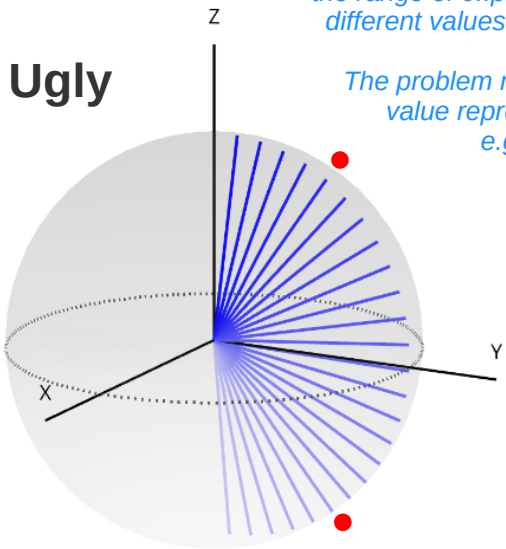


Positive values are represented by blue vectors and negative values by red vectors.

This example shows encoding values wrapping around the range of 0..pi of the Bloch sphere, ensuring that different values are represented by unique angles.

The problem may arise if we may have the same value represented by two distinct amplitudes, e.g. values of sin(x) represented by x

(violates principle 2)

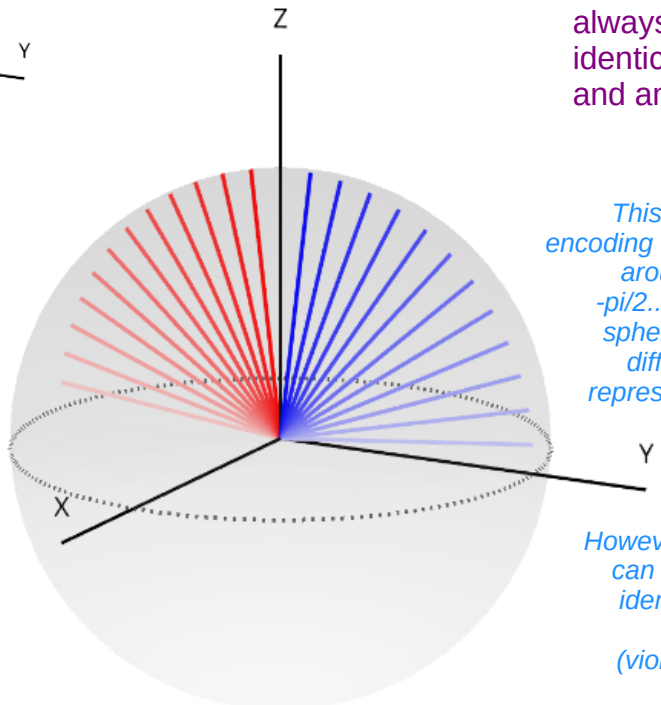


Special care must be taken in those cases where the values form some kind of symmetry or repetition. e.g. in a case values x and 2pi+x effectively represent the same angular position and their encoding should also be identical.

The red dot represents the same value which on two occasions maps onto two different angles and thus two amplitudes.

Two principles of quantum data encoding:

1) distinct data values should map onto distinct angles and amplitudes
2) the same data values should always map into identical angles and amplitudes



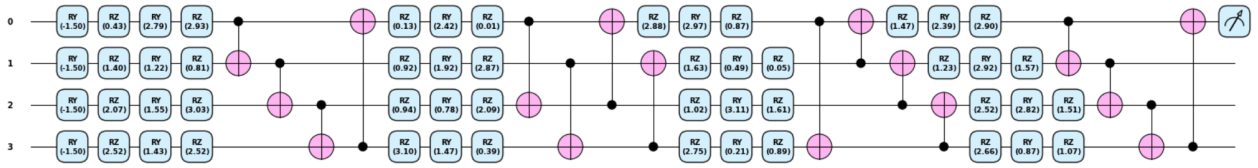
This example shows encoding values wrapping around the range of -pi/2..pi/2 of the Bloch sphere, ensuring that different values are represented by unique angles.

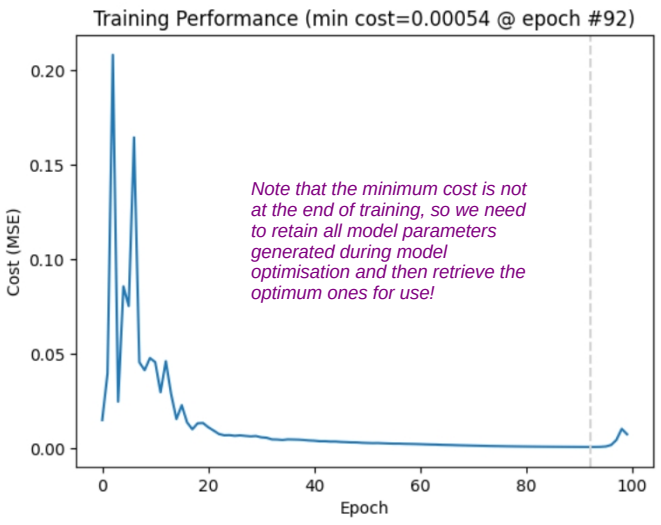
However, some angles can be measured as identical amplitudes

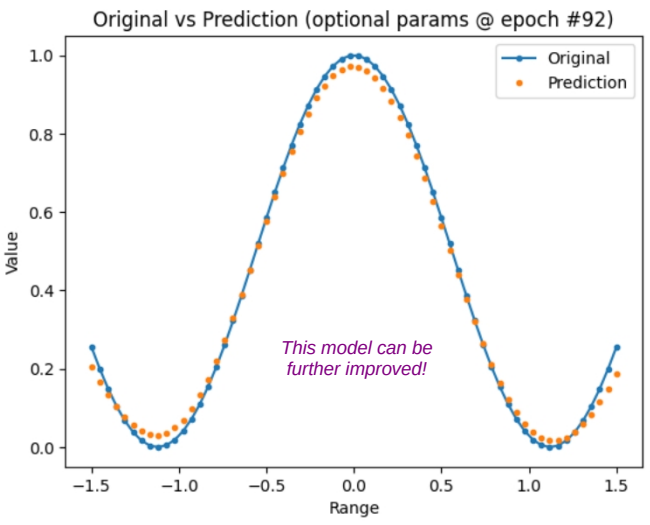
(violates principle 1)

Improved model training
Lessons learnt

The same circuit as the best circuit used with the first dataset








```
Training completed: epochs=100, min cost=0.00054 @ 92, time=029 secs
```

Corrected / improved the model:

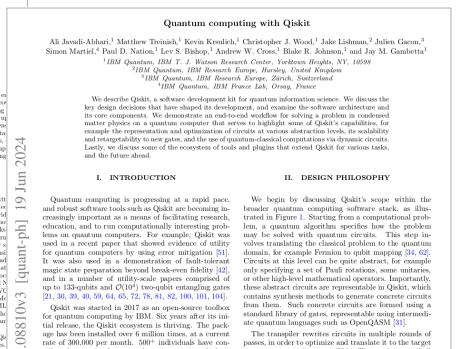
- Corrected data encoding to map into the valid rotation range
- Introduced a simple data reuploading (across wires)
- Logging model parameters during optimisation, so that we could use the best rather than last model

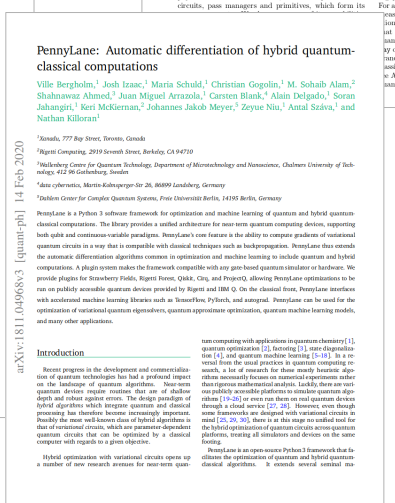
Improved code / presentation quality:

- Implemented the cost function and a circuit as closures (no globals)
- Created a model shape function to match model factory function
(no more code replication)
- Used PennyLane standard functions for creating encoding and entangling blocks
- Increased flexibility by separating circuit and qnode creation
- Parameterised all aspects of models creation and training
(no hard-coded constants)
- Changed circuit text-based drawing to graphical representation (mpl)
- Improved presentation of the chart plotting training costs

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
- There exist a variety of different methods of quantum encoding
- There exist a variety of different ways of interpreting state measurements
- PennyLane is the QML of differentiable programming (everything is a function)
- When training a quantum model log costs and model parameters - you may need both later on (e.g. scoring training / testing models)

- Simple data may lead to the deceitful feeling of success!
- Bad data encoding is the harbinger of a good model's death!
- What are the principles of quality PennyLane code?
  - Encapsulate models and cost functions in closures
    Note: closures are differentiable, class instances are not!
  - Keep model shape function near the model definition functions
  - Data reuploading often greatly improves model accuracy
  - Use PennyLane standard functions (great way to cut your effort)
  - Parameterise all aspects of models creation and training

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

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/

https://github.com/ironfrown/pennylane_intro_hands_on/