Objective of this session:

To undertake a brave, hands-on, and independent exploration of hybrid quantum-classical machine learning, with the minimum of assistance!

QML hybrid quantum-classical models

Structure and processes of a hybrid PyTorch / PennyLane model
Brief introduction to hybrid reservoir computing

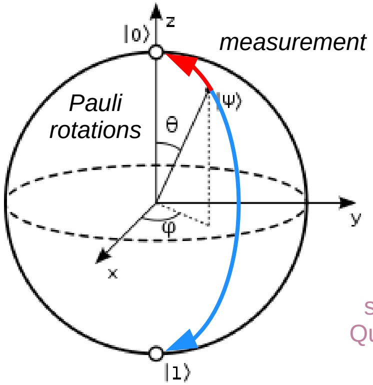
PennyLane demo for some!
Dive into action for others!

Note that in these practical tasks you will battle errors!

Quantum Machine Learning

Hybrid quantum-classical models
Advanced session with minimum instruction

Jacob L. Cybulski
Enquanted, Melbourne, Australia



We will assume some knowledge of Quantum Computing ML and Python

12-13 April 2025, QPoland, Fundacja Quantum AI

Hybrid
Quantum-Classical Models

Hybrid quantum-classical models in PyTorch is easy, as neural networks with PennyLane quantum layers.

Example structure of such a model is to the right.

Data preparation, training and scoring are identical for both. Just keep it in mind that a quantum model must receive data in the correct format for quantum encoding and then produce data in the format understood by the next neural net layer.

Ensure that the quantum layer actually adds value to the classical neural network models.

Test hybrid model performance against pure classical models and pure quantum models.

A good example where a quantum model actually adds value to the classical solution is in *hybrid reservoir computing*.

As quantum models transform input data into large-dimensional Hilbert space, they are able to perform the task needed by classical reservoir models, i.e. to increase data dimensionality to assist linear separability of information.

```python
##### Custom PyTorch/PennyLane hybrid model for logistic regression
class LogisticRegression(torch.nn.Module):

    ### build the constructor
    def __init__(self, sim, n_wires, n_layers=1, shots=None):
        super().__init__()

        self.sim = sim
        self.n_wires = n_wires
        self.n_layers = n_layers
        self.shots = shots

        # A very simple hybrid model
        qmodel = self.qlayer()              # Quantum layer
        lin5 = torch.nn.Linear(2, 1)        # Classical layer
        layers = [qmodel, lin5]
        self.model = torch.nn.Sequential(*layers)

    ### PyTorch layer around the PennyLane model
    def qlayer(self):

        # Specify a device
        dev = qml.device(self.sim, wires=self.n_wires, shots=self.shots)

        # Define the quantum model and its circuit (or node, save it for later)
        model_pl = qmodel(self.n_wires)
        self.model_qc = qml.QNode(model_pl, dev, interface='torch')

        # Define the shape of the model weight parameters
        # Note that the name "weights" must match the param name defined in function
        # "model_pl" which in our case is _qmodel(inputs, weights)
        weights_shapes = {"weights": qshape(self.n_wires, n_layers=self.n_layers)}

        # Turn the circuit into a Torch-compatible quantum layer
        qlayer = qml.qnn.TorchLayer(self.model_qc, weight_shapes=weights_shapes)
        return qlayer

    ### Return the quantum model circuit
    def qmodel_qc(self):
        return self.model_qc

    ### Make predictions
    def forward(self, x):
        y_pred = self.model(x)
        return y_pred
```

PyTorch Neural Net mixing classical layers with quantum PennyLane layers

Here is a utility function to return the quantum model used in neural network structure (e.g. for drawing)

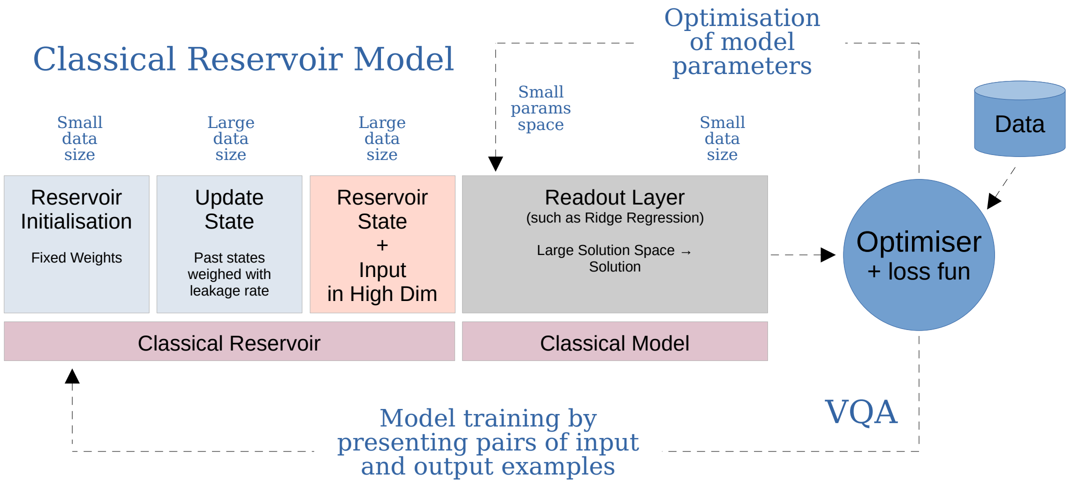
Reservoir computing

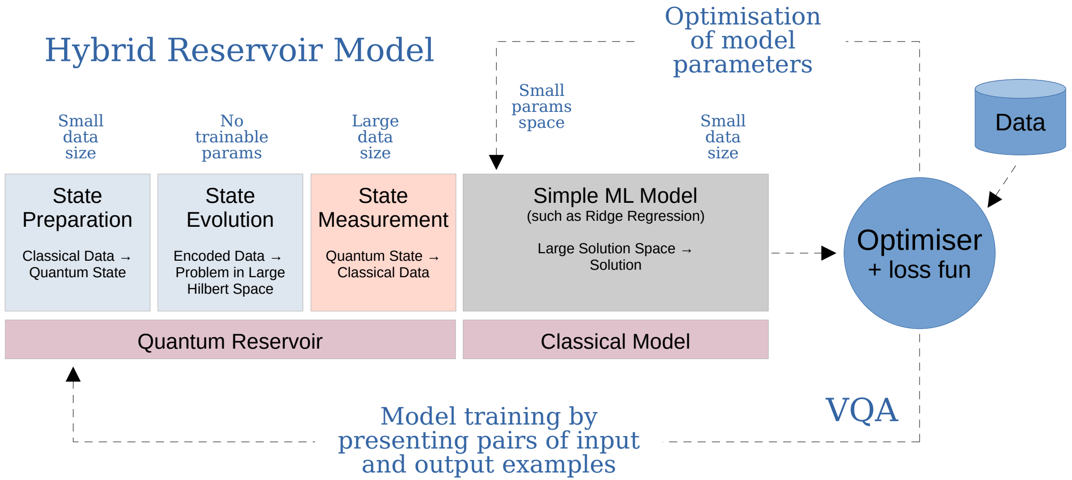
for temporal data (series and signals)

Classical RC models are derived from recurrent neural networks. They are especially useful when working with *temporal data*. Reservoir computing utilises a reservoir – a large sparse neural network of randomly initialised and fixed weights, which transform input into a *higher-dimensional space*. In high-dimensional space, *data can be easily separated* (classified) by using a simple linear model (readout layer), such a ridge regression.

The reservoir should be able to *echo* or *retain information* about past inputs for a short period, making it suitable for processing sequential data. However, the influence of past input on the reservoir state should fade away over time – *memory leakage*.

Applications include time-series forecasting, speech recognition and video analysis, control of robots or autonomous vehicles, as well as, predicting weather patterns and stock markets.





Hybrid Quantum-Classical Design

Hybrid quantum reservoir models consist of two parts:

- *quantum reservoir model* (echo and leakage)
- *classical readout model* (e.g. ridge regression)

Quantum reservoir parameters are sparse, random and fixed (no need for training).

The aim of the quantum model is for input to gain in dimensionality, to increase linear separability.

The classical readout is very simple and easy to train.

Hybrid reservoirs are highly efficient and accurate.

Thank you!

The great finale!

Any reflections?
Any requests?
Any questions?

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