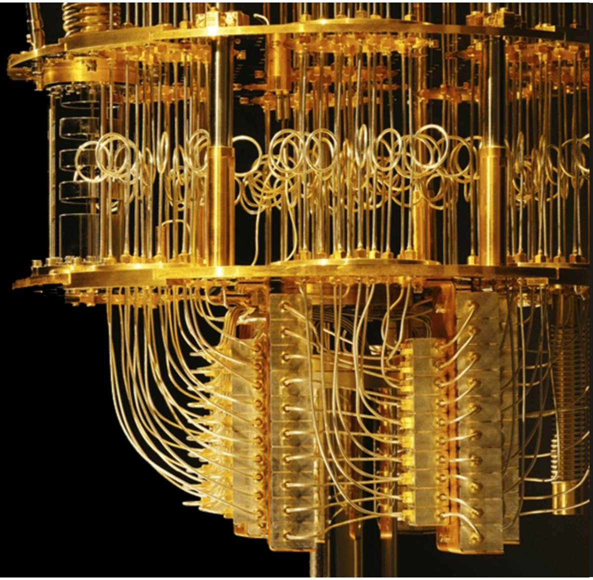


IBM quantum computer

Quantum computing is modern magic
Quantum machine learning turns data into magic

Key Concepts in Quantum Time Series Analysis (QTSA)
An Introduction and Preliminary Research Results

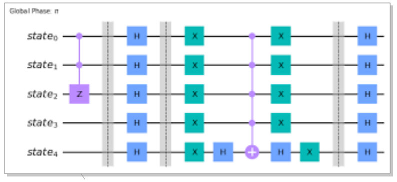
Jacob L. Cybulski
School of IT, SEBE, Deakin University

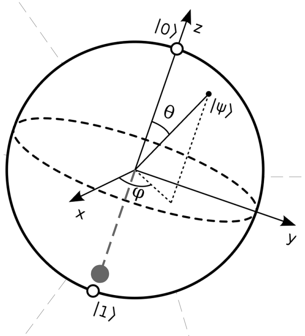
29 September 2022

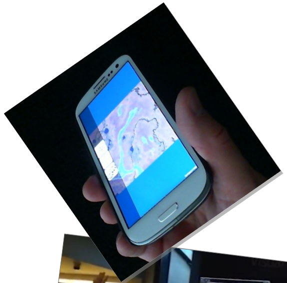
Presenter

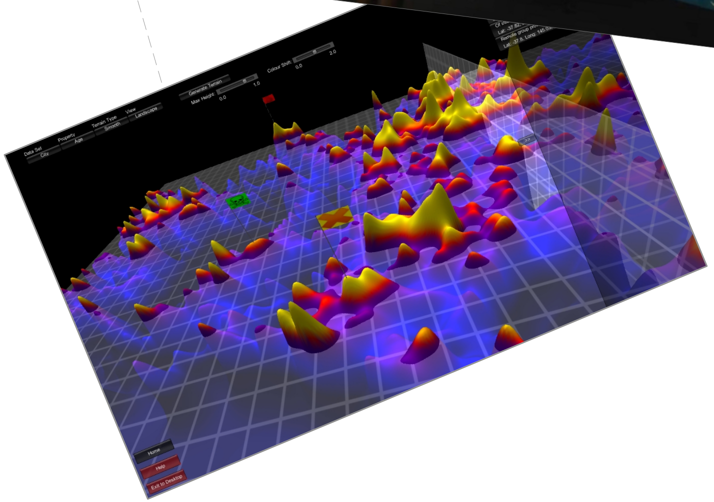
Jacob Cybulski
jacob.cybulski@deakin.edu.au

Honorary A/Prof
In Quantum Computing
School of IT, SEBE
Deakin University
Melbourne, Australia







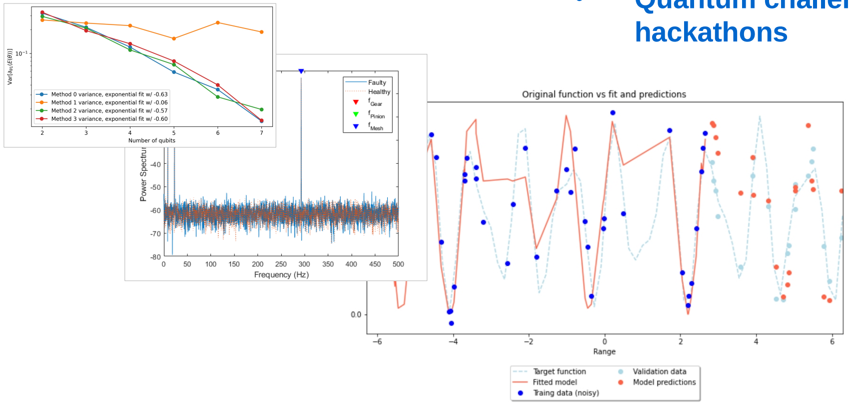


Research

- Quantum computing
- Quantum machine learning
- Quantum time series analysis and anomaly detection
- Classical machine learning
- Data visualisation

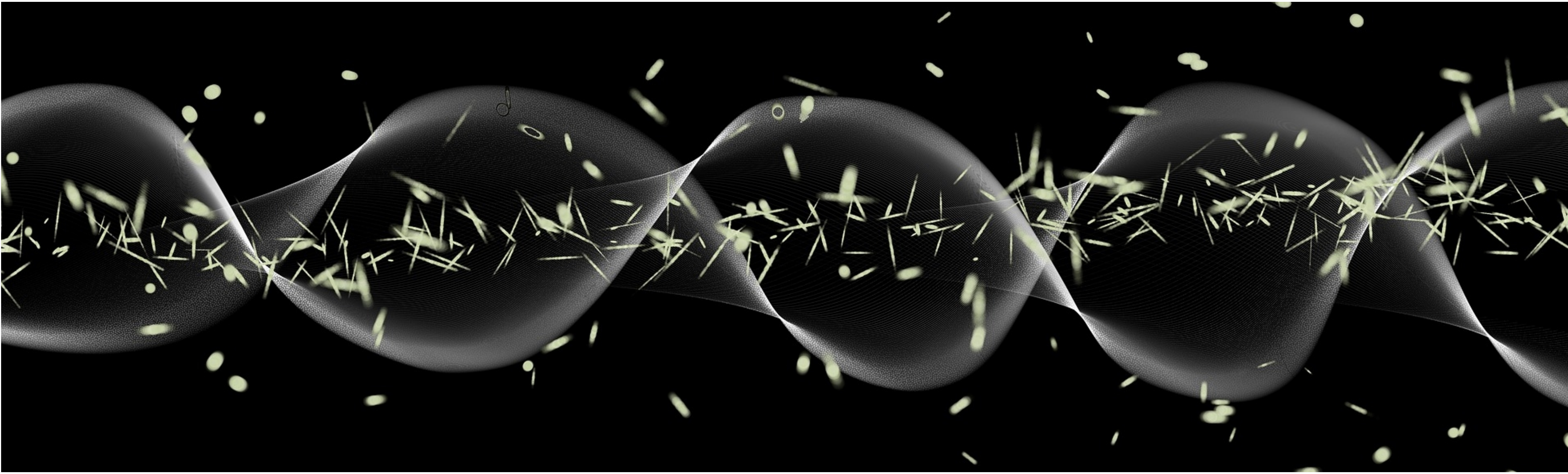
Personal

- Recreational cycling
- Reading science and Sci-Fi
- Quantum challenges and hackathons



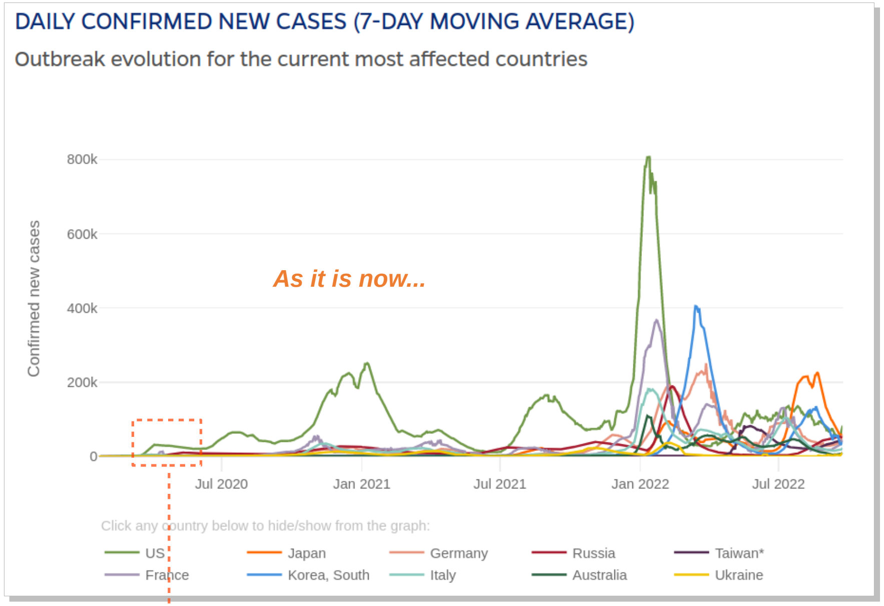
Classical TS Methods

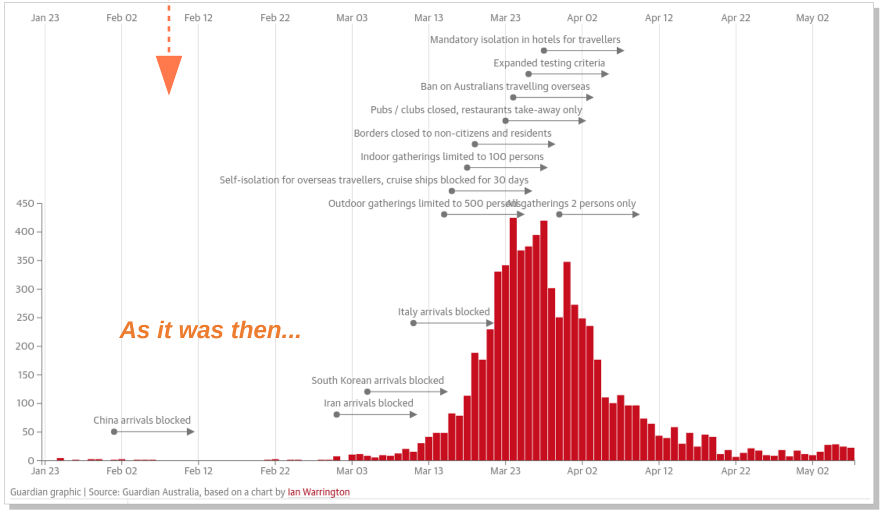
Key concepts in classical time series analysis



Analysis of the past
Prediction of the future

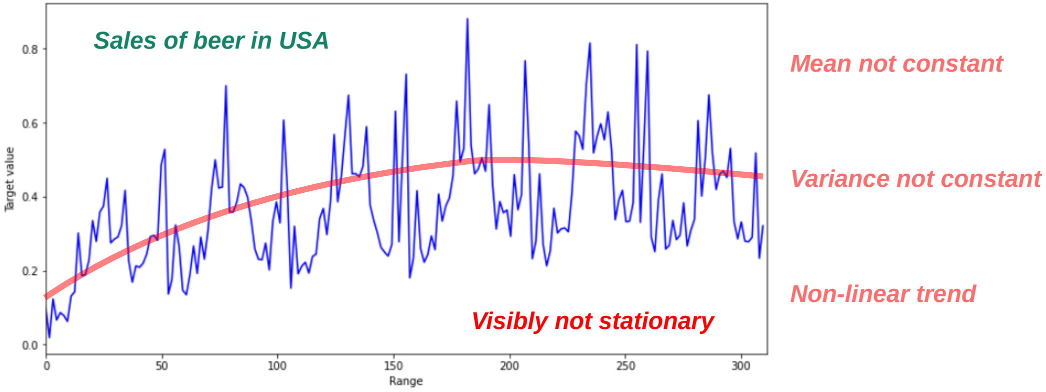
- We are bombarded daily with volumes of time-based information, often in the form of data points in time, in other words - a *time series*
- Time series visualisations are used to inform experts, influence government policies and shape public opinion (e.g. about COVID)
- Time series analysis aims to *identify patterns* in collected historical data and to create *forecasts* of what data is likely to be collected in the future
- Sample applications include heart monitoring, weather forecasts, fault detection in rotating machinery, etc.
- Times series analysis and forecasting is an established and trusted discipline, with excellent tools and highly efficient methods
- Organisations that rely on time-based information are in the pursuit of more efficient or more effective time series analysis
- *Quantum time series is a possible approach to time series analysis and forecasting*





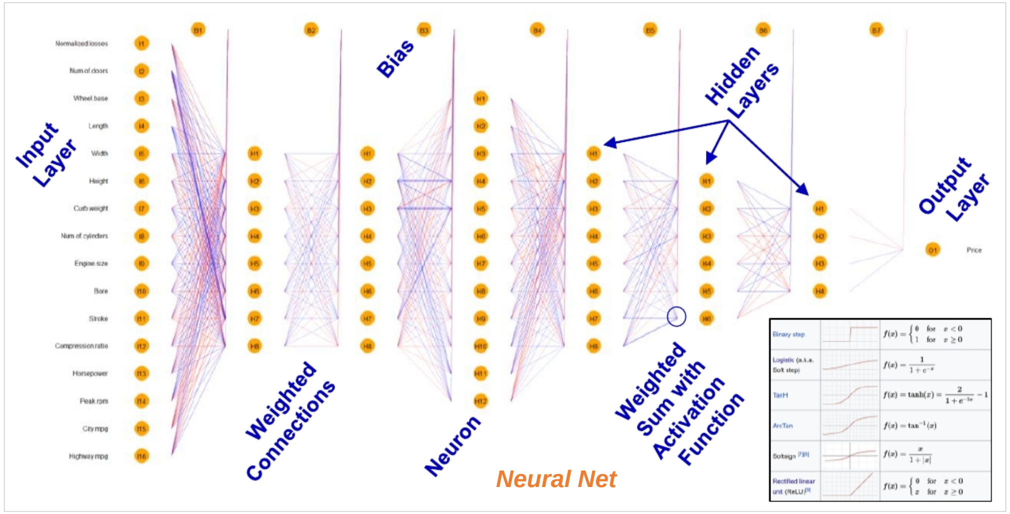
Key concepts in time series analysis

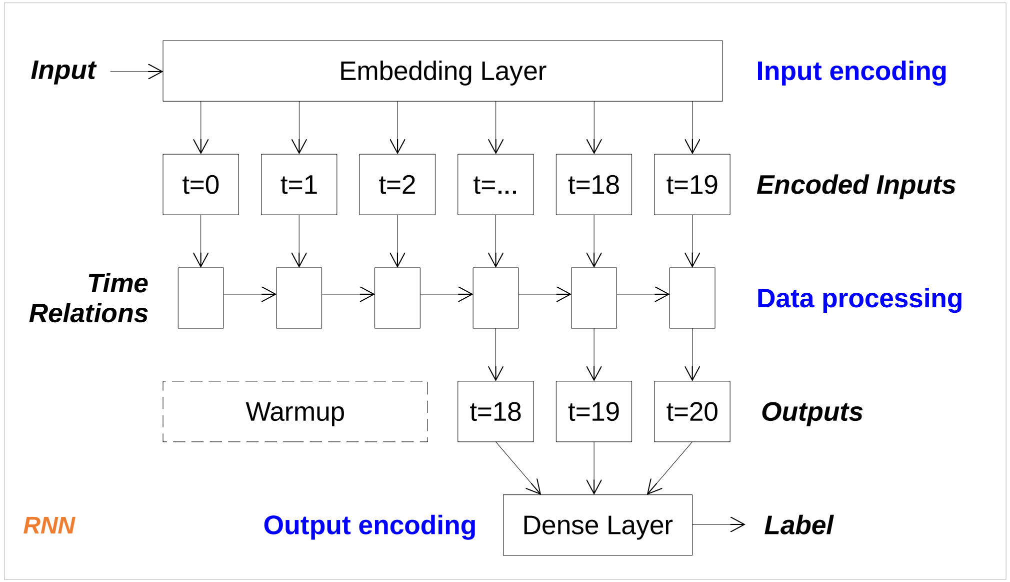
- As with any data set, time series needs some *pre-processing* for its effective use
- Time series must have an *index*
  - a time-stamp sequencing the series
- It is often assumed that index is a key, i.e. *index values are unique*
- Time series needs to be *ordered* by its index
- Time series will also have some *time-dependent attributes* to be modelled
- Time series can be *univariate* or *multivariate*, depending on whether a single or multiple attributes are being investigated
- *Missing indeces* and their dependent attributes may need to be imputed (e.g. interpolated)
- A series can be defined over non-time entities, e.g. a landscape line or a DNA sequence
- Index needs to be of appropriate *granularity*, e.g. years, months, weeks, days, hours, etc.
- Attributes need to be *aggregated* to the required index granularity
- Time signal often shows *seasonality* in data, i.e. a regular repeating pattern
- With aggregation and smoothing seasonality can be removed and *trends* visually identified
- Majority of forecasting methods require *time-series to be stationary*, i.e. its mean, variance and auto-correlation are constant
- Time series analysis needs *data storage*



Neural Nets for Time Series Analysis

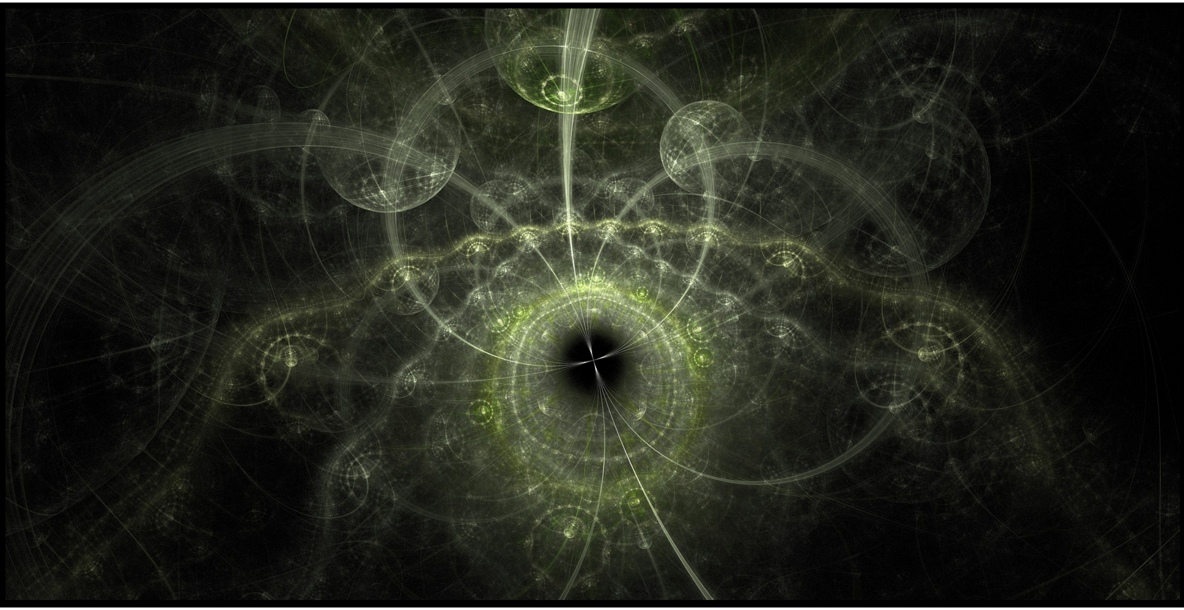
- The simplest neural networks, such as Multi-Layer Perceptrons (MLPs), map numeric inputs into numeric or categorical outputs via layers of “neurons”, interconnected by weighed links, and calculating weighted sums with non-linearity
- The weights of neural links are trained within an optimisation process, such as gradient descent, by matching the calculated vs expected outputs
- Some types of deep neural networks can be trained for time series analysis, including: forecasting, classification and clustering, e.g.
  - Recurrent Neural Networks (RNN)
  - Long Short-Term Memory (LSTM) nets
  - Gated Recurrent Units (GRU) nets
- Unlike MLPs, networks such as RNN, LSTM and GRU are able to retain and rely on memory of the past training data
- Neural networks and RNNs are similar in their structure to quantum solutions (circuits)





Fundamental Quantum Concepts

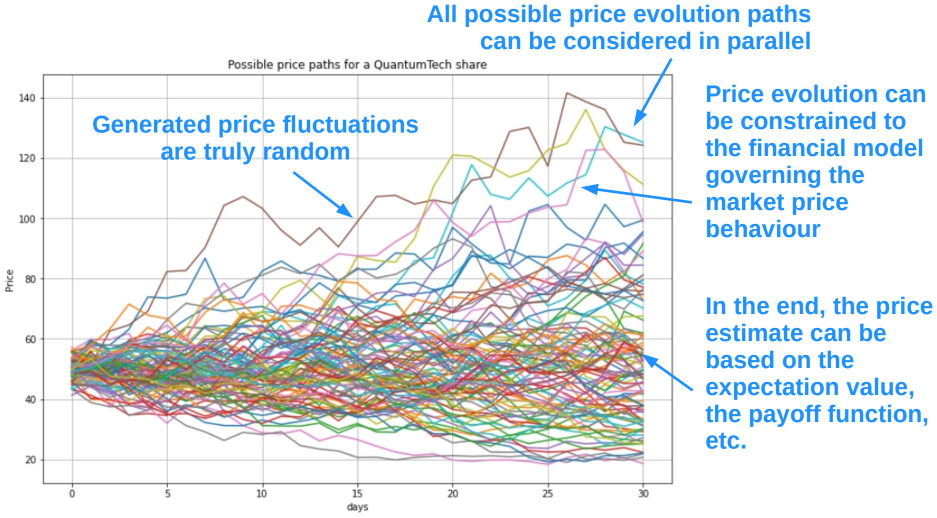
Approach, qubits and circuits, process and principles



Quantum Computing
Approach and applications

- *Quantum computing allows information processing to be accomplished by utilising the behaviour of matter and light on the atomic and subatomic scale*
- Quantum computing aims at solving problems in many disciplines, e.g.
  - *natural science*, such as calculation of molecular energy or protein folding;
  - *finance*, such as portfolio optimisation, pricing of financial options or credit risk assessment;
  - *optimisation*, such as in vehicle routing or energy distribution using several quantum-enhanced optimisation techniques;
  - *machine learning*, featuring many general purpose algorithms, such as neural networks or kernel methods.

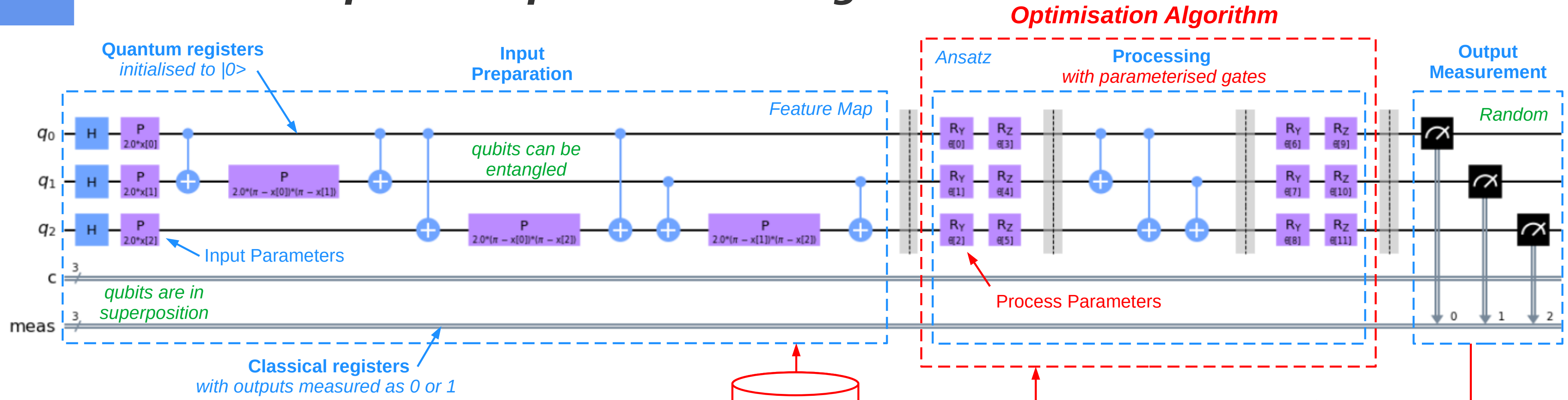
- Quantum applications can demonstrate their advantage over classical solutions by relying on the following features of quantum systems:
  - True randomness of observable results *(measurement of quantum states)*
  - Pursuing alternative decisions concurrently *(superposition of quantum states)*
  - Controlling parallel choices with constraints *(entanglement of quantum elements)*
- An example where all three principles are applied – financial option price prediction

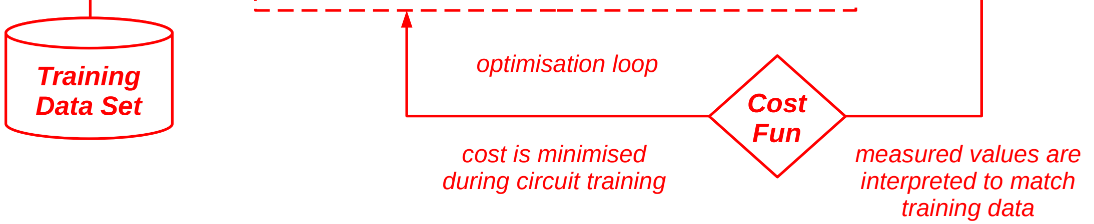


Quantum Machine Learning

Process of quantum problem-solving

Qubit is the most important quantum tech concept. It is a unit of quantum information. It is also a device able to manipulate a single unit of such information.





Quantum circuit is the model of quantum computation. It identifies qubits, their initialisation to some input states, a sequence of quantum gates performing operations on qubits (such as rotation or entanglement), and measurements allowing observation of qubit states, on circuit execution.

Quantum circuits are static - new data requires new circuit. However, it is possible to create “variational” circuits, which are templates with parametrised gates, e.g. P, Ry and Rz varying degrees of rotation, which can be optimised using some ML algorithm.

To find optimum circuit parameters, the circuit is repeatedly executed and its outputs measured. Outputs are then compared against the expected values using a cost function, so the optimiser could determine new values for the process parameters.

How qubits work
In a simplified way!

Qubit is often "implemented" as a single elementary particle, e.g. electron or photon

Qubit represents a state of such a particle, e.g. an electron spin (up or down) or photon's linear polarisation (horizontal or vertical)

Qubits are in a state of *superposition* of some *basis states*, so the electron spin is not just up or down but a combination of these basis states, e.g.

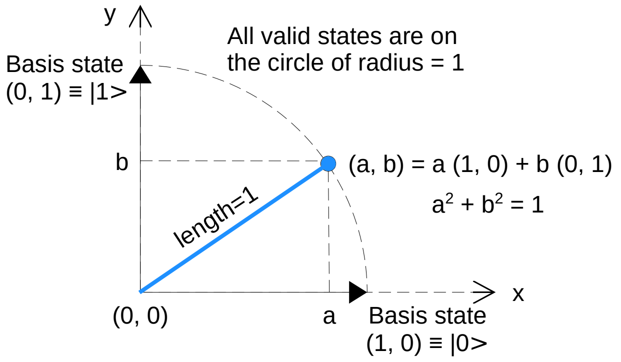
$$\frac{\sqrt{3}}{2} \times \underline{up} + \frac{1}{2} \times \underline{down}$$

When we *measure* the qubit, its state collapses probabilistically into one of the basis states up or down, measured as simple values, e.g.

- 0 / 1 for up or down for electrons, and
- 0 / 1 for horizontal or vertical for photons.

Mathematically a qubit state can be represented as a vector (a point) in space of all possible states.

The *qubit state space* has its own coordinate system, defined by the basis vectors, which are orthogonal unit vectors (of length=1), for example in 2D these could be vectors (1, 0) and (0, 1), denoted as |0> and |1>.



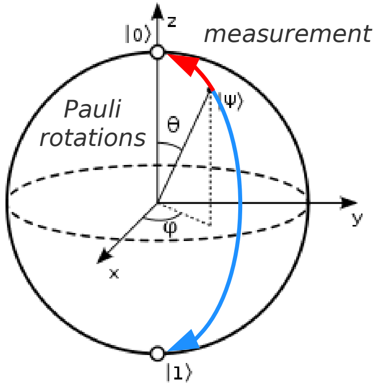
|0> and |1> are vectors, not values 0 and 1, which is what we observe on qubit measurement

Qubit measurement returns 0 or 1, but repeated measurements provide expectation values of observing 0 or 1

As we can see, any vector (a, b) is a (linear) combination of the basis vectors (1, 0) and (0, 1) – we call this *superposition*.

In reality, this is more complex as qubit states are described by two *complex numbers*, each with real and imaginary parts, making it 4 reals.

With a clever "projection" trick we can depict the qubit state in 3D (not 4D), e.g. as a *Bloch sphere*.



Experiments
Data Encoding and Data Analysis

Ways of utilising time series data in quantum system computation

- *Quantum systems have no memory!*
- *Quantum circuits take no data!*

- The only way of obtaining and retaining information in a quantum system is via:
  - *structure of a quantum circuit* (as done in variational methods)
  - *states of quantum computation* (as done in adiabatic optimisation)
- In this presentation we will focus on the first option – the *variational methods*
- In variational time series analysis, the key concerns are:
  - *data encoding strategy*
  - *circuit optimisation strategy*

There are many different quantum data encoding / state preparation methods:

- *basis encoding*, with qubits acting as bits in the encoded number (int) to be processed further in the circuit
- *angle encoding*, where qubit rotation (real) represents the value of data
- *amplitude encoding*, where each data point is encoded as expectation value of the measured circuit (real), usually no further data processing is present
- Others: QuAM, QRAM, Qsample, ...

Can be further interpreted

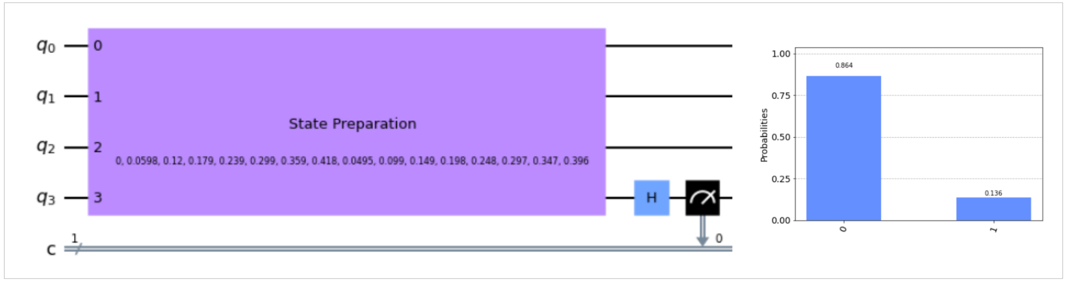
We will explain these approaches by demonstration of:

- *Quantum regression (function fitting)*
- *Quantum Fourier transform (function fitting)*
- *Quantum neural networks (pattern detect.)*

Schuld, Maria, and Francesco Petruccione. *Machine Learning with Quantum Computers. 2nd ed.* 2021 edition. Springer, 2021. https://link.springer.com/book/10.1007/978-3-030-83098-4.

Variational quantum linear regression

Function Fitting



We are trying to find *a* and *b* to satisfy a linear equation

$$\vec{y} = a\vec{x} + b$$

We will encode a normalised vector *y* as a quantum state $|y\rangle$

$$|y\rangle = \frac{1}{C_y}\vec{y} = \frac{1}{C_y}(a\vec{x} + b)$$

We will identify *a* and *b* in the optimisation process, which considers a sequence of states:

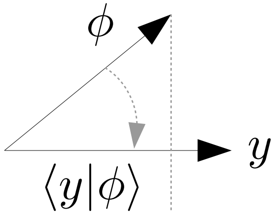
$$|\phi\rangle = \frac{1}{C_\phi}(a\vec{x} + b)$$

$$|\phi_0\rangle, |\phi_1\rangle, \ldots, |\phi_n\rangle$$

Starting with $|\phi_0\rangle = |\phi\rangle\,(a_0, b_0)$

The optimisation will search for such *a* and *b*, and thus $|\phi\rangle$ to minimise the cost function C_p which tries to maximise the similarity of $|y\rangle$ and $|\phi\rangle$

$$C_p = (1 - \langle y|\phi\rangle)^2$$



What's remaining is to create a quantum circuit able to calculate $\langle y|\phi\rangle$ based on pairs *a* and *b*, so that the cost function could drive the optimisation process

Can be adapted for fitting higher order polynomials

The required circuit will be built into the cost function. It will rely on the *amplitude encoding* of sample data, which ensures that measured expectation values of the composite qubit states corresponds to data values

We encode normalised *ax+b* in qubits q_0 and q_1, and normalised *y* in qubits q_2 and q_3. These encoded values act as constraints on the calculation. The Hadamard gate *H* allows measuring the expectation value, which implies the value of the inner product of interest *(for explanation, see ref: Qiskit 2020)*

Bravo-Prieto, Carlos, Ryan LaRose, Marco Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J. Coles. "Variational Quantum Linear Solver." *ArXiv Preprint* ArXiv:1909.05820, 2019.

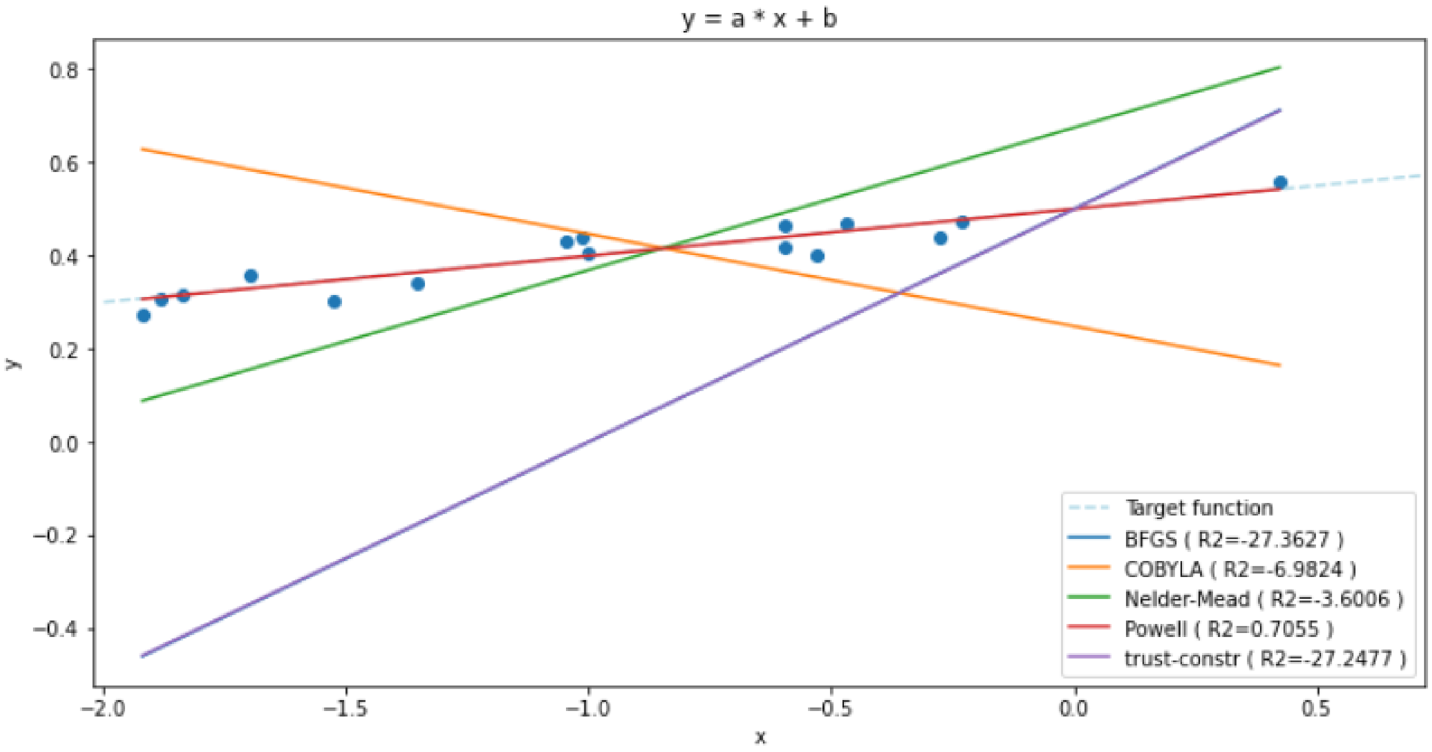
Qiskit. "Variational Quantum Regression", in *Learn Quantum Computation Using Qiskit. Textbook*, 2020. https://qiskit.org/textbook/ch-demos/variational-quantum-regression.html

Experiment R1: Target - Line Fit

Linear Regression

data: samples_train=16, samples_valid=0
x0=[0.5, 0.5], max_iter=200

```
BFGS         R2:   -27.3627
BFGS         MAPE: 0.9512

COBYLA       R2:   -6.9824
COBYLA       MAPE: 0.4836

Nelder-Mead  R2:   -3.6006
Nelder-Mead  MAPE: 0.3436

Powell       R2:   0.7055
Powell       MAPE: 0.0901

trust-constr R2:   -27.2477
trust-constr MAPE: 0.9492
```

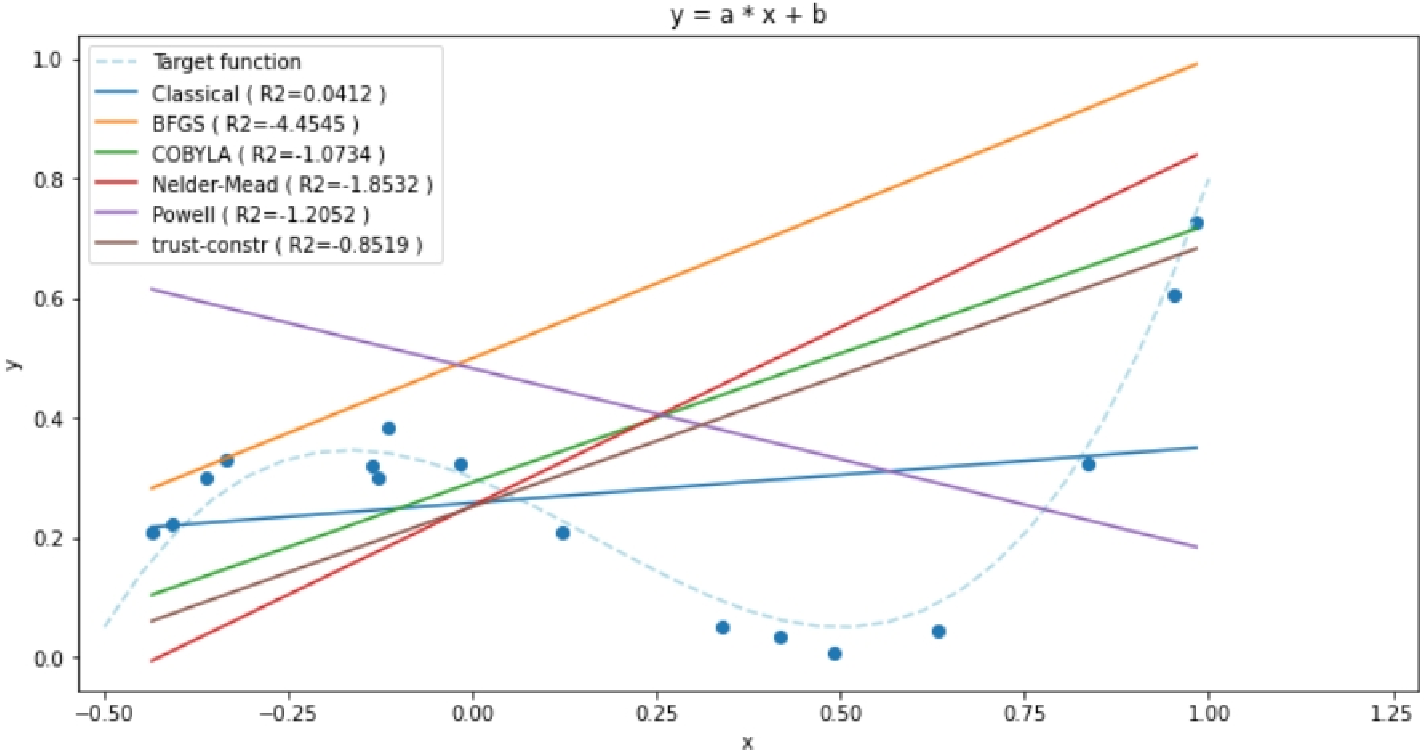


Here data is only 16 points
This data fitting was surprisingly slow!
Different optimisers give vastly different results, regardless of their hyper-parameters!

Experiment R2: Target - Poly3 Fit

data: samples_train=16, samples_valid=0
x0=[random, ...], max_iter=200

```
Classical    R2:   0.0412
Classical    MAPE: 3.573

BFGS         R2:   -4.4545
BFGS         MAPE: 9.554

COBYLA       R2:   -1.0734
COBYLA       MAPE: 6.3691

Nelder-Mead  R2:   -1.8532
Nelder-Mead  MAPE: 7.1835

Powell       R2:   -1.2052
Powell       MAPE: 3.9021

trust-constr R2:   -0.8519
trust-constr MAPE: 5.9791
```



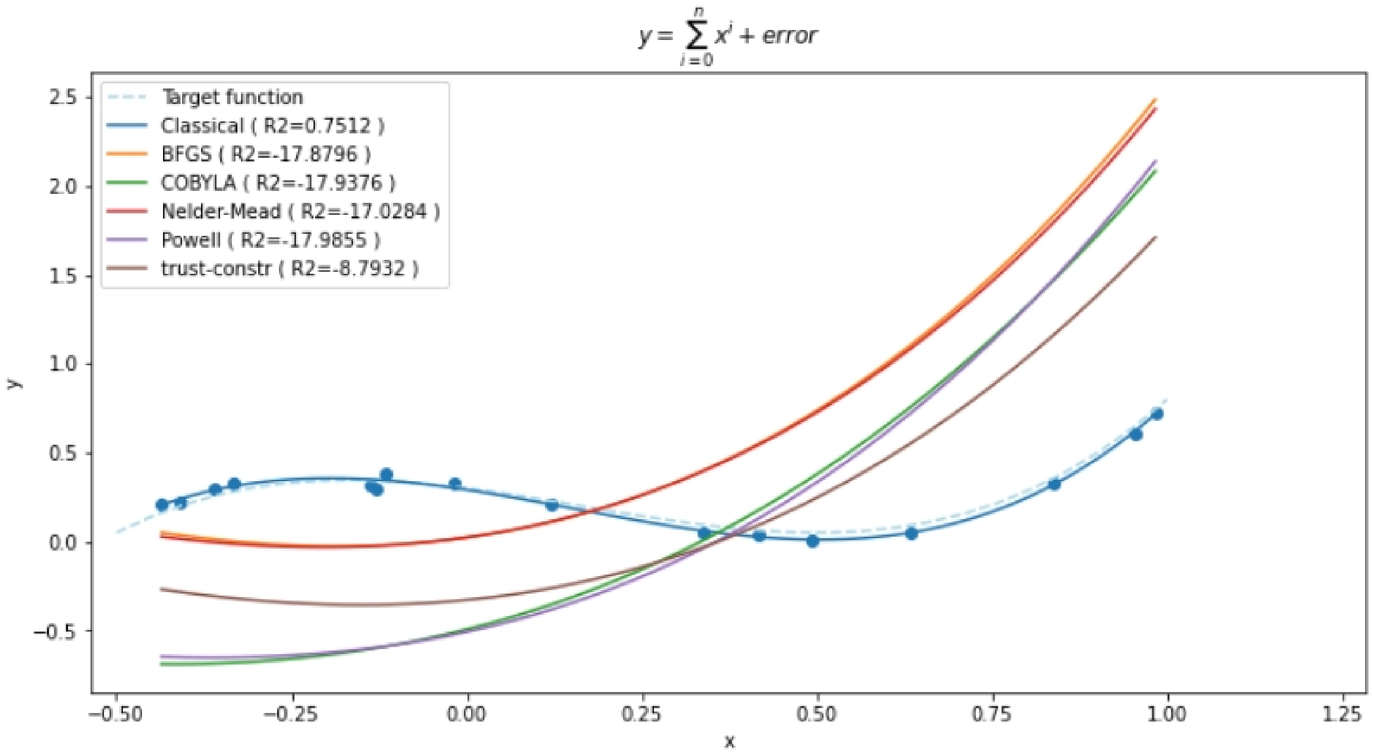
Here data is only 16 points
This data fitting was surprisingly slow!
Different optimisers give vastly different results!

Experiment R3: Target - Poly3 Fit

data: samples_train=16, samples_valid=0
x0=[random, ...], max_iter=200

Higher Order Poly Fit

```
Classical    R2:   0.7512
Classical    MAPE: 0.5323

BFGS         R2:   -17.8796
BFGS         MAPE: 12.075

COBYLA       R2:   -17.9376
COBYLA       MAPE: 8.857

Nelder-Mead  R2:   -17.0284
Nelder-Mead  MAPE: 11.8813

Powell       R2:   -17.9855
Powell       MAPE: 8.4221

trust-constr R2:   -8.7932
trust-constr MAPE: 6.247
```



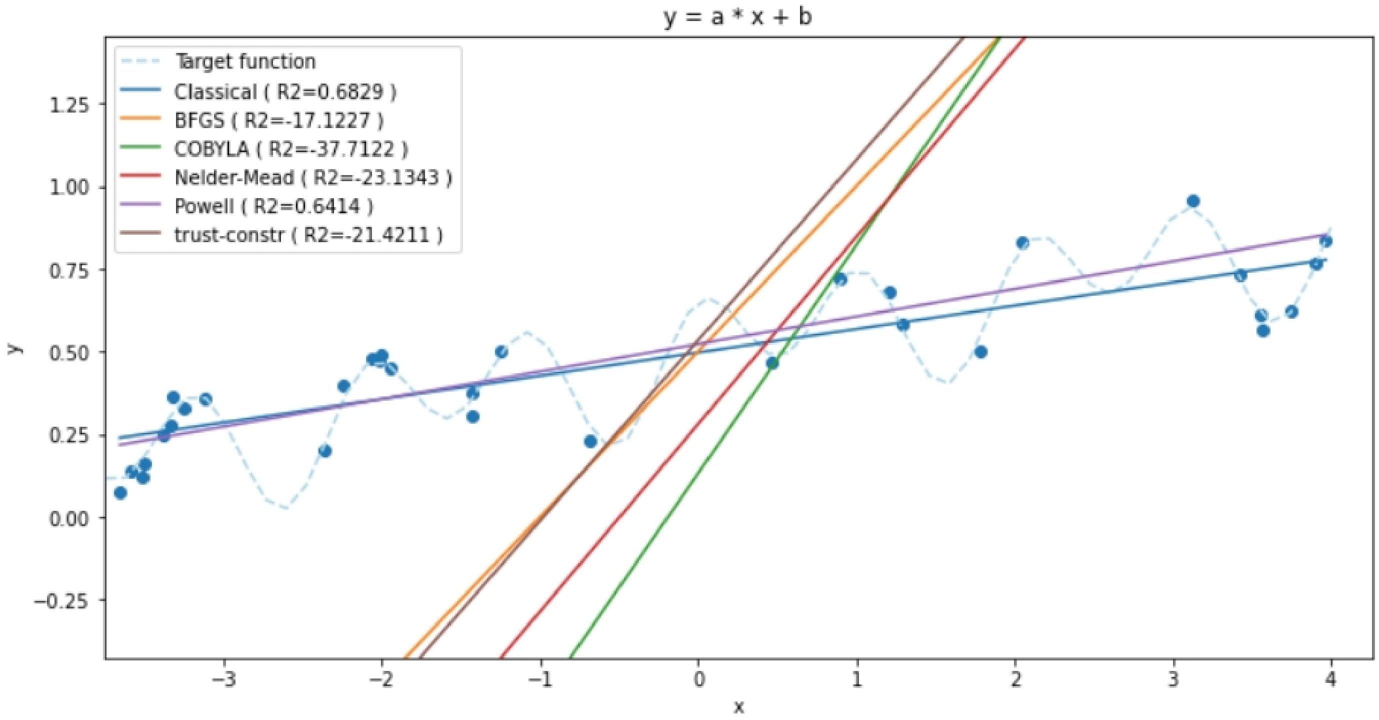
Here data is only 16 points
This data fitting was surprisingly slow!
Different optimisers give vastly different results!

Experiment R4: Target_Trig_trend

data: samples_train=32, samples_valid=0
x0=[random, ...], max_iter=200

Linear Regression

```
Classical    R2:   0.6829
Classical    MAPE: 0.3618

BFGS         R2:   -17.1227
BFGS         MAPE: 2.8504

COBYLA       R2:   -37.7122
COBYLA       MAPE: 4.6102

Nelder-Mead  R2:   -23.1343
Nelder-Mead  MAPE: 3.5516

Powell       R2:   0.6414
Powell       MAPE: 0.3647

trust-constr R2:   -21.4211
trust-constr MAPE: 3.1285
```



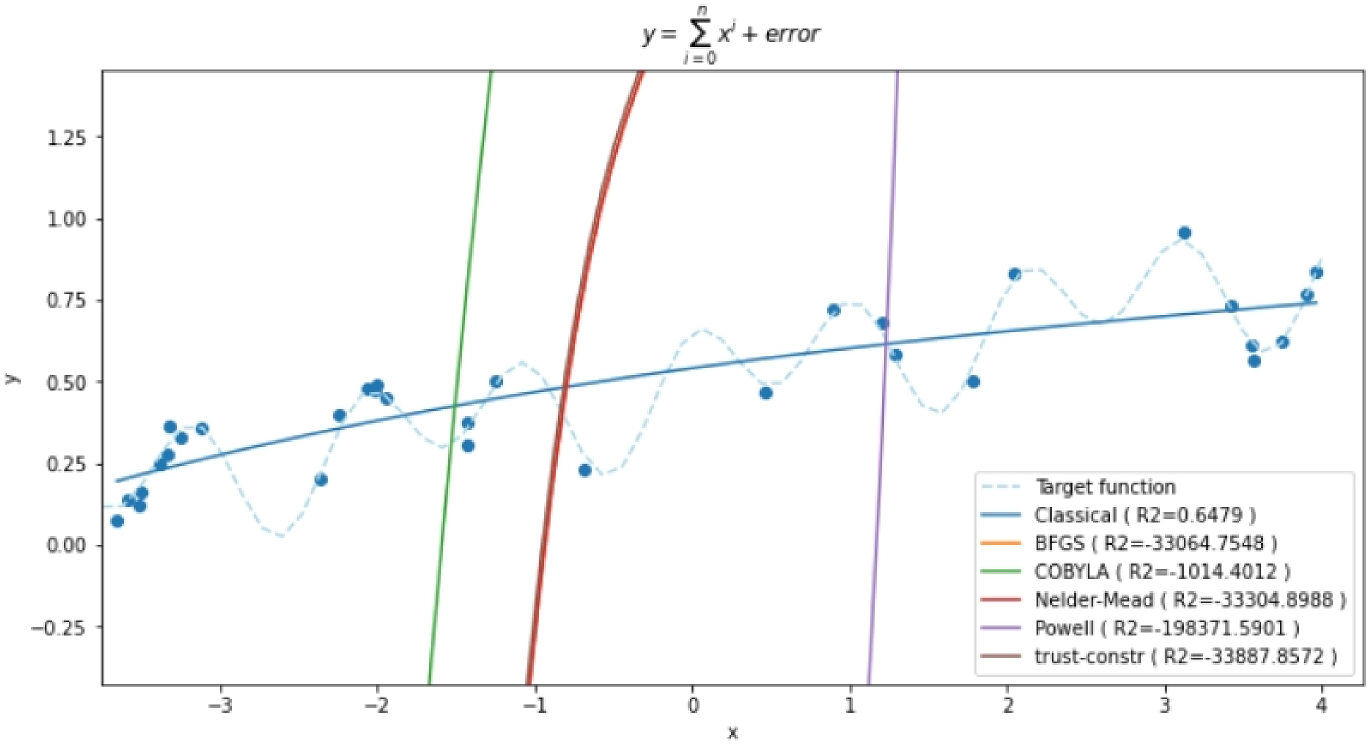
Here data is 32 points
The fit is terrible, all except for one optimisers have failed!

Experiment R5: Target_Trig_trend

Higher Order Poly Fit

data: samples_train=32, samples_valid=0
x0=[random, ...], max_iter=200

```
Classical    R2:   0.6479
Classical    MAPE: 0.3614

BFGS         R2:   -33064.7548
BFGS         MAPE: 95.3883

COBYLA       R2:   -1014.4012
COBYLA       MAPE: 26.5104

Nelder-Mead  R2:   -33304.8988
Nelder-Mead  MAPE: 95.76

Powell       R2:   -198371.5901
Powell       MAPE: 325.245

trust-constr R2:   -33887.8572
trust-constr MAPE: 95.7221
```



Here data is 32 points
The fit is terrible, all optimisers have failed!

Variational quantum linear regression Reflections

- Variational quantum regression can only be used to fit linear data (with some noise)
- Its training convergence is highly sensitive to the optimisation strategy and the optimiser’s hyper-parameters
- While the approach taken can be easily adapted to fitting higher order polynomials, only certain types of functions fit successfully (mainly those appearing in publications)
- As compared with classical methods of linear/polynomial fitting, experiments with variational quantum regression indicate the adopted quantum regression approach is not promising
- It is worth noting that other, more recent methods such as QSVT (Quantum Singular Value Decomposition), can assist fitting any function (or data) with higher-order polynomials

Variational quantum Fourier transforms

Schuld, Maria, Ryan Sweke, and Johannes Jakob Meyer. "The Effect of Data Encoding on the Expressive Power of Variational Quantum Machine Learning Models." *Physical Review A* 103, no. 3 (March 24, 2021)

Pérez-Salinas, Adrián, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. “Data Re-Uploading for a Universal Quantum Classifier.” Quantum 4 (Feb 6, 2020): 226.

PennyLane. "Quantum models as Fourier series", 2021. https://pennylane.ai/qml/demos/tutorial_expressivity_fourier_series.html

We consider a quantum model of the following form, which takes 1D data

$$f_\theta = \langle 0|U^\dagger(x,\theta) M U(x,\theta)|0\rangle$$

where M is a measurement observable, *U(x, θ)* is a variational quantum circuit that encodes data and depends on params *θ*

The circuit consists of *n* layers, each with encoding block $S_n(x)$, which is a Pauli rotation gate, and a trainable block $W_n(\theta_n)$

$$S_n(x) = e^{inx}$$

We can now rewrite f_θ as as a Fourier-like sum of “frequency” components.

$$f_\theta(x) = \sum_{n\in\Omega} W_n(\theta_n) S_n(x)$$

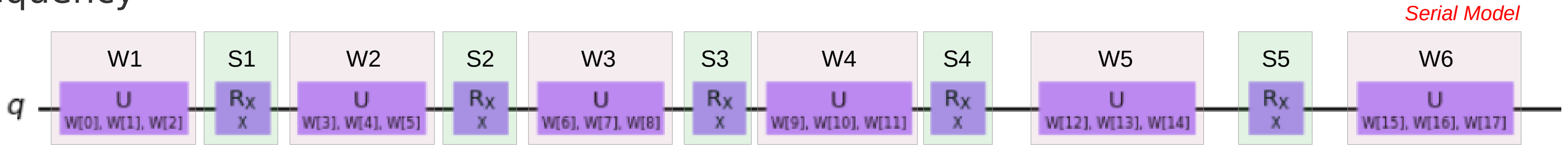
The circuit is structured as a *series* of $W_n(\theta_n)$ $S_n(x)$ layers over a single qubit, with data repeatedly re-uploaded

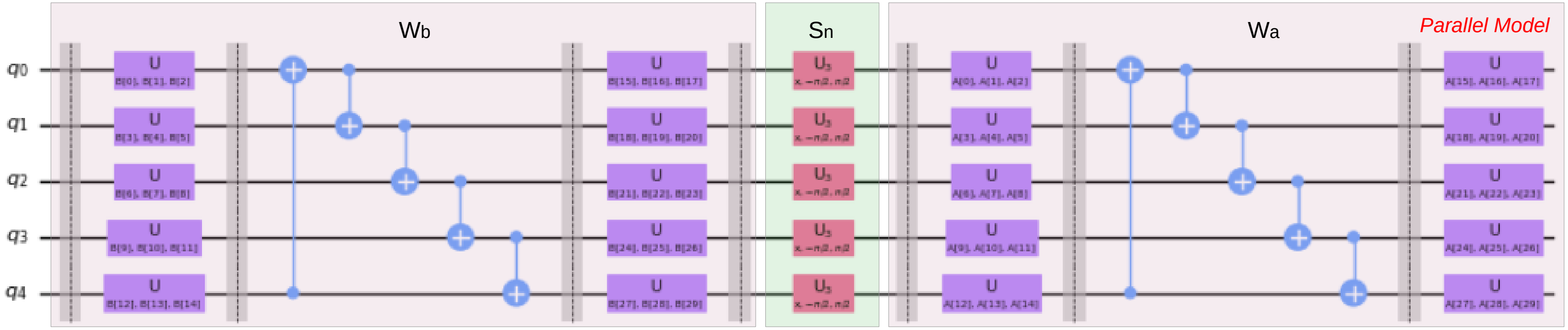
Alternatively, $S_n(x)$ blocks can be arranged in *parallel*, with $W_{a/b}(\theta)$ blocks before and after, over multiple qubits

The circuit parameters can then be trained by the ML optimiser of choice

Th components are determined by $S_n(x)$ “frequencies” and $W_n(\theta_n)$ coefficients

Re-uploading of $S_n(x)$ allows to vary the “frequencies” by accumulating rotations $W_n(\theta_n)$.

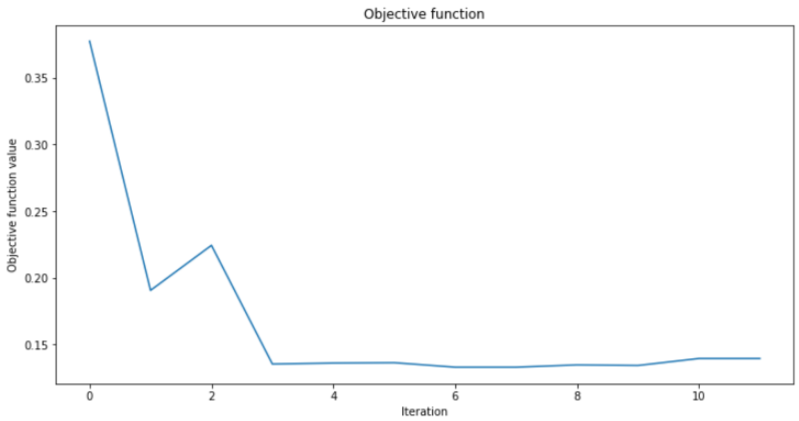




Experiment F1: Fit for Sin function

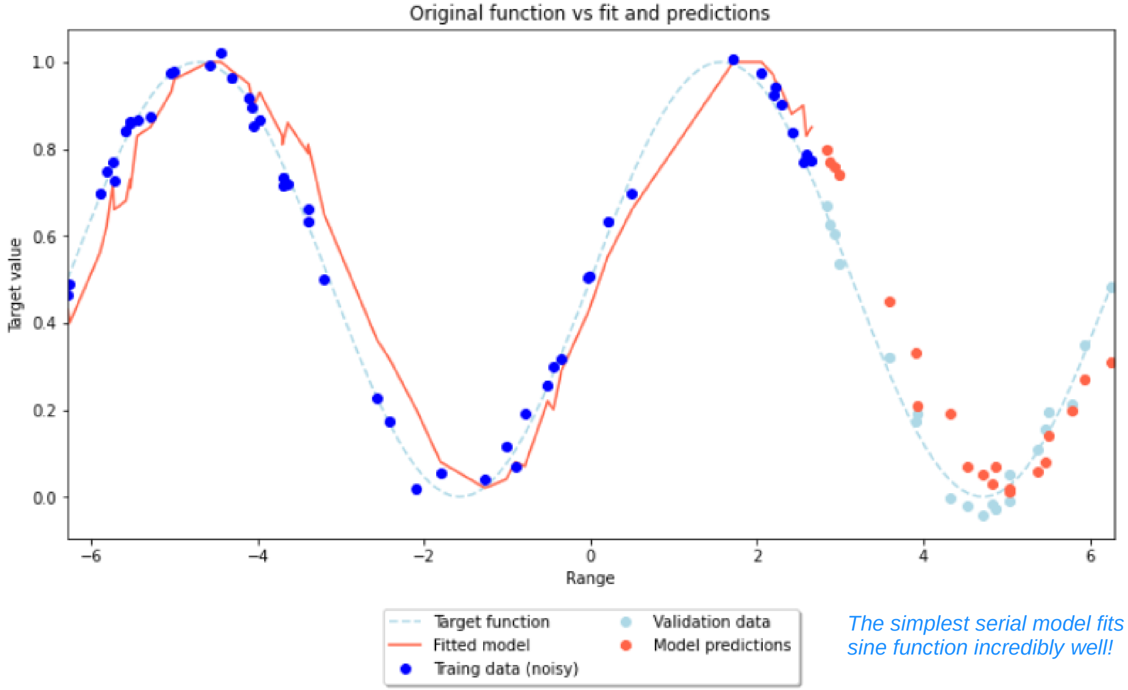
Serial Model

sin(): samples_train=50, samples_valid=20
layers=1, optimizer=L_BFGS_B(maxiter=16)

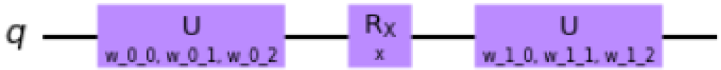



```
Minimum objective function value: (6, 0.1330683974385614)
```

```
R2 for training data vs ground truth:     0.9918349791907765

R2 for pred vs training data:             0.9077718132939956
MAPE for pred vs training data:           0.33358423582057184

R2 for pred vs validation data:           0.8008426173608819
MAPE for pred vs validation data:         7.663155696226305
```



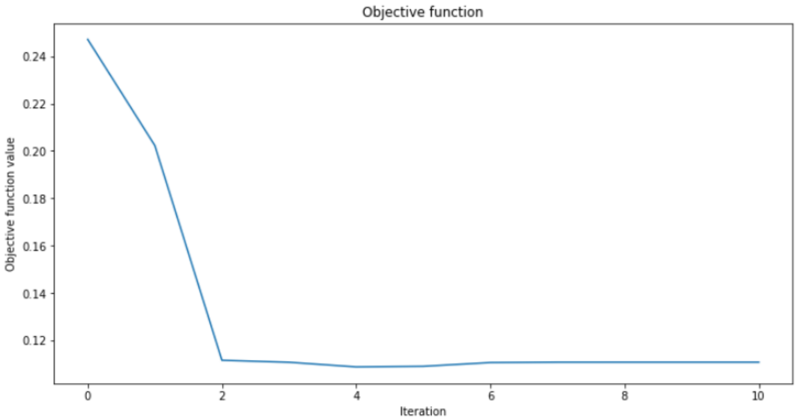
The simplest serial model fits the sine function incredibly well!



Experiment F2: Fit for Sin function

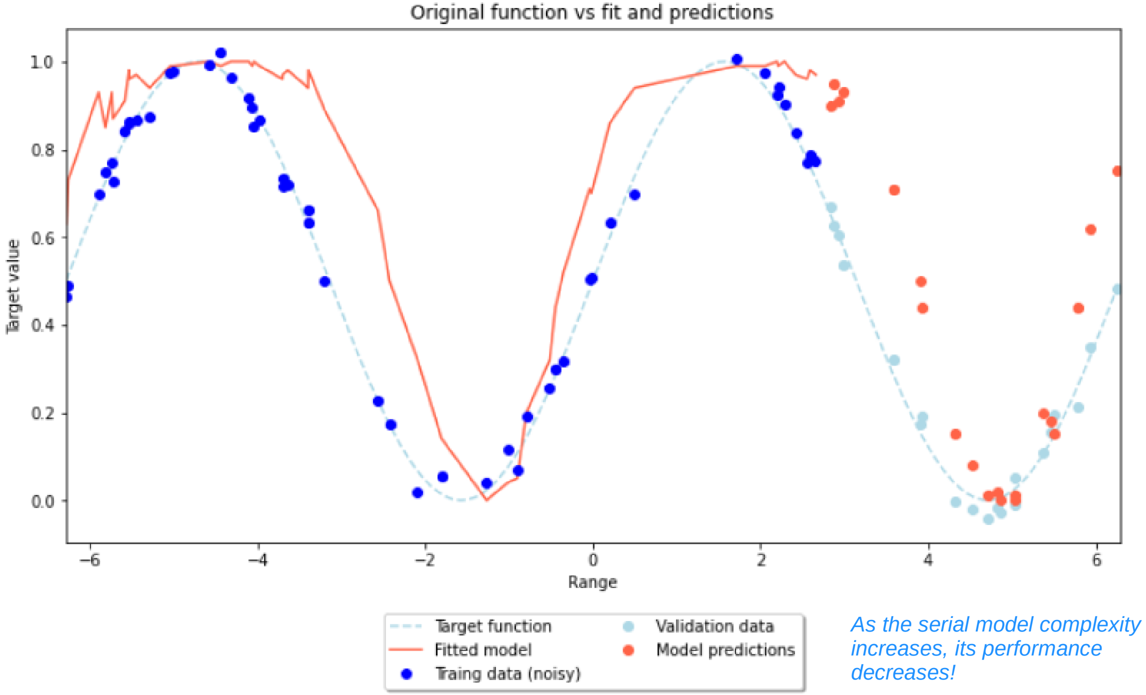
Serial Model

sin(): samples_train=50, samples_valid=20
layers=5, optimizer=L_BFGS_B(maxiter=16)




```
Minimum objective function value: (4, 0.10879146887817771)
```

```
R2 for training data vs ground truth:     0.99183497919077 65

R2 for pred vs training data:             0.6099357156304572
MAPE for pred vs training data:           0.6889922735024059

R2 for pred vs validation data:           0.05423997067201114
MAPE for pred vs validation data:         14.09823617905607
```

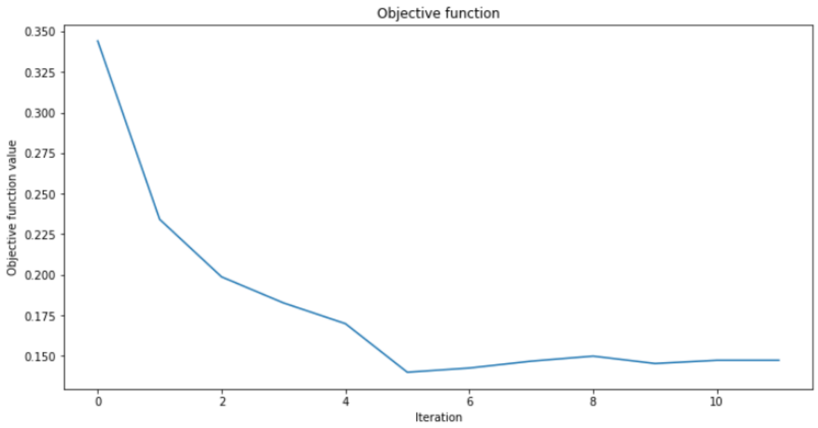


As the serial model complexity increases, its performance decreases!

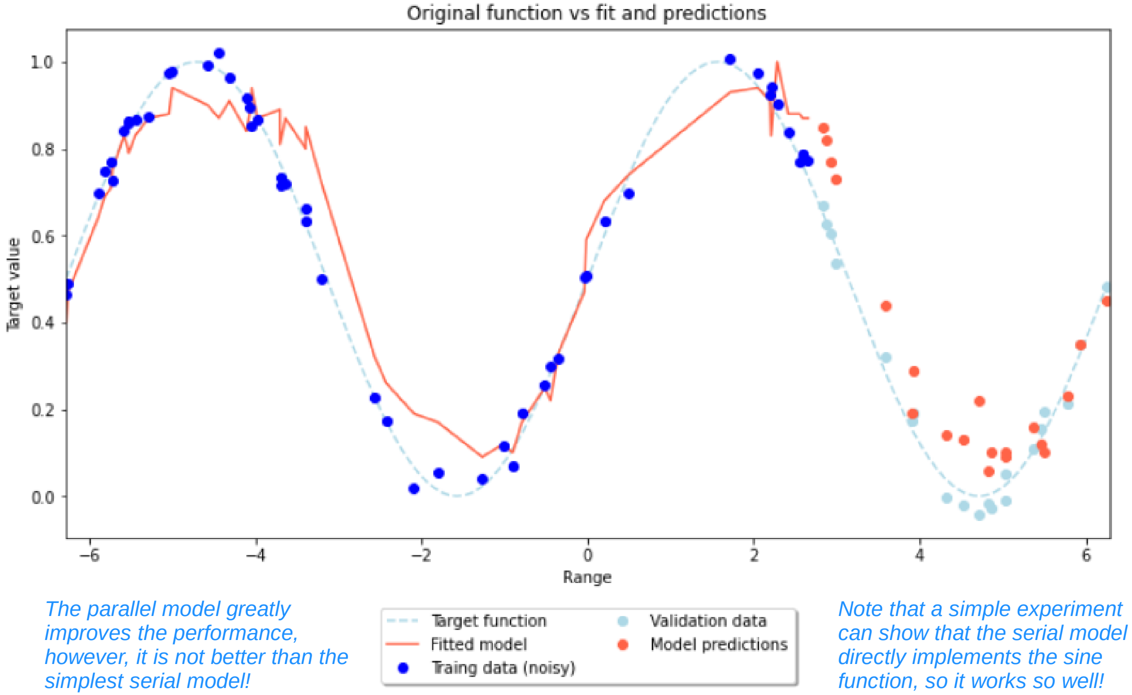
Experiment F5: Fit for Sin function

Parallel Model

sin(): samples_train=50, samples_valid=20
qubits=5, layers=2, interpretation=parity, optimizer=L_BFGS_B(maxiter=15)




```
Minimum objective function value: (5, 0.13994586134911505)
```




```
R2 for training data vs ground truth:     0.9918349791907765

R2 for pred vs training data:             0.8874290876885176
MAPE for pred vs training data:           0.427966185294369

R2 for pred vs validation data:           0.6294743588196634
MAPE for pred vs validation data:         11.225205765589482
```

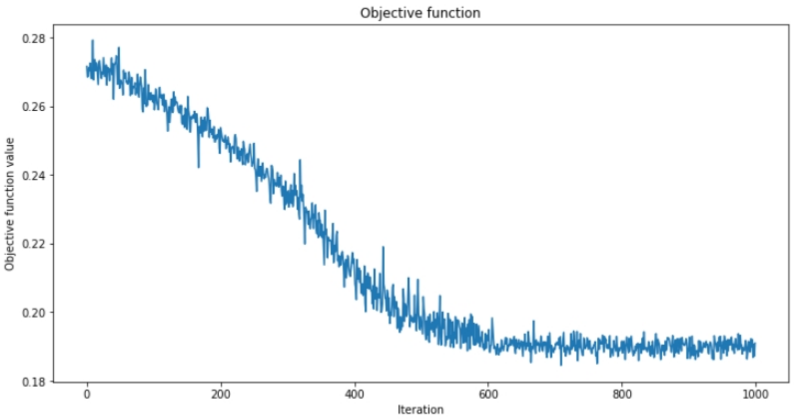
The parallel model greatly improves the performance, however, it is not better than the simplest serial model!

Note that a simple experiment can show that the serial model directly implements the sine function, so it works so well!

Experiment F7: Fit for 2-Sins function

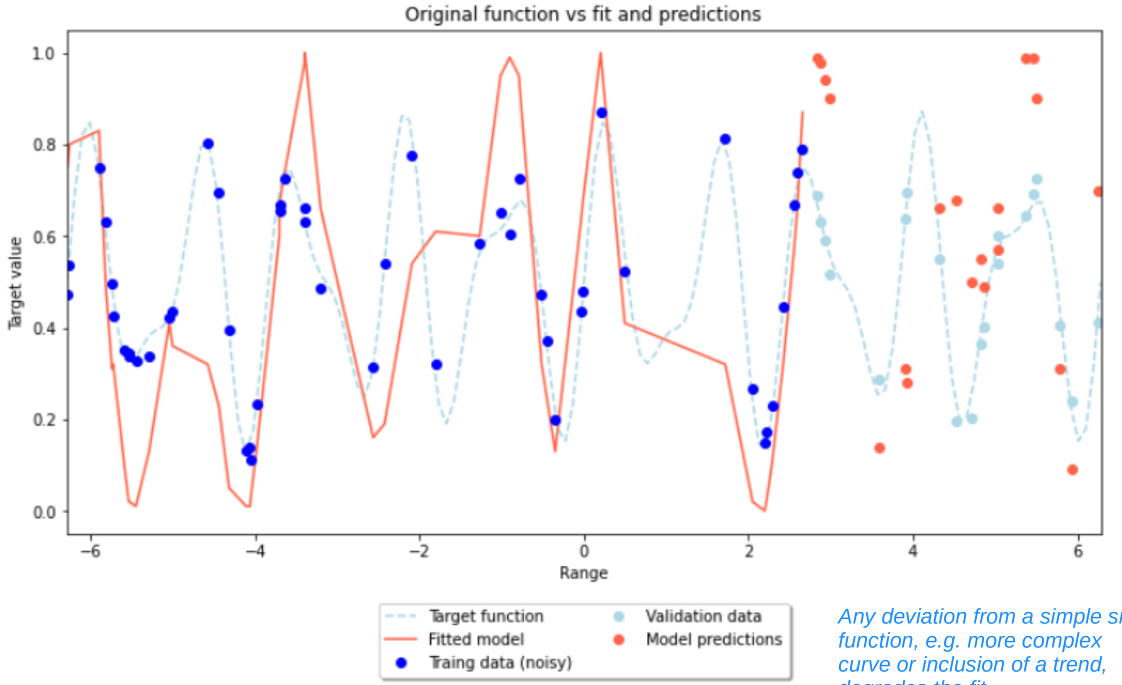
Serial Model

2-sins(): samples_train=50, samples_valid=20
layers=15, optimizer=NELDER_MEAD()

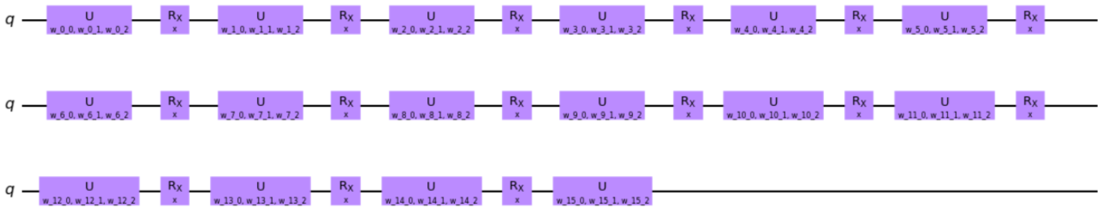



```
Minimum objective function value: (709, 0.1844968894314135)
```

```
R2 for training data vs ground truth:     0.9827475536429731

R2 for pred vs training data:             -0.19097777701942054
MAPE for pred vs training data:           0.48021284794775143

R2 for pred vs validation data:           -1.6245309914112311
MAPE for pred vs validation data:         0.5864867791493156
```



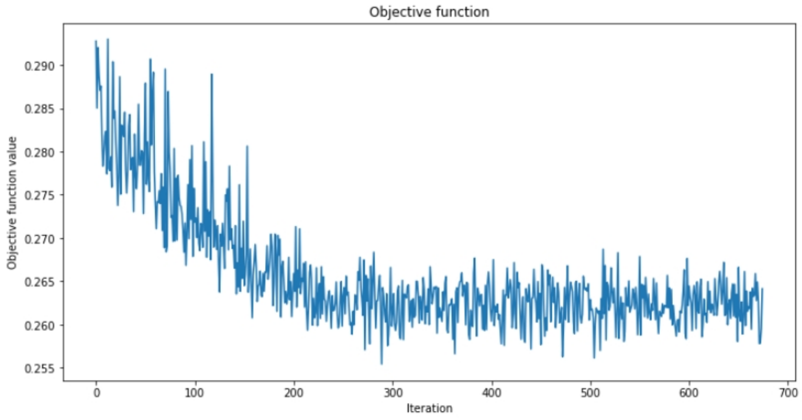
Any deviation from a simple sine function, e.g. more complex curve or inclusion of a trend, degrades the fit.



Experiment F8: Fit for 2-Sins function

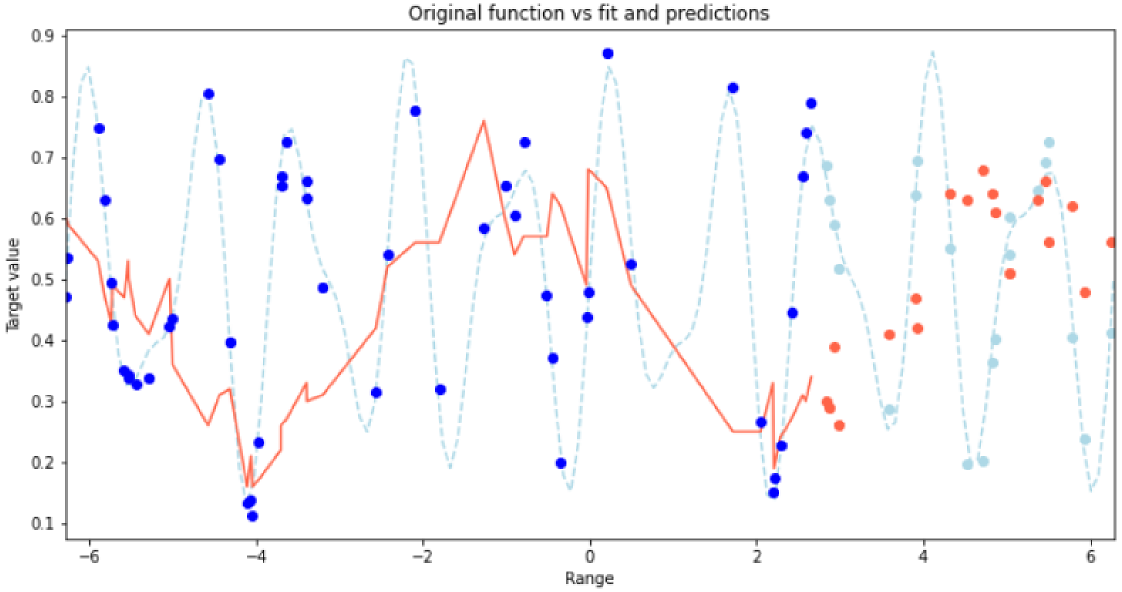
Parallel Model

2-sins(): samples_train=50, samples_valid=20
qubits=5, layers=2, optimizer=COBYLA()

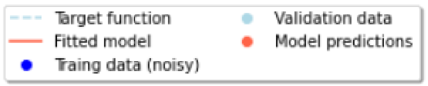



```
Minimum objective function value: (289, 0.2554413317536141)
```

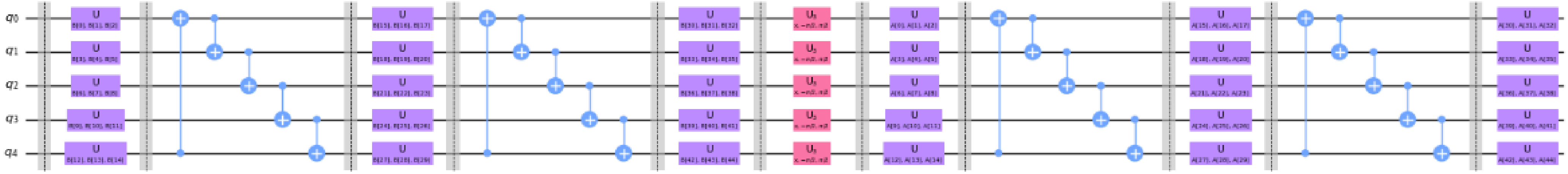
```
R2 for training data vs ground truth:     0.9827475536429731

R2 for pred vs training data:             -0.37705567940830365
MAPE for pred vs training data:           0.42001786798711505

R2 for pred vs validation data:           -0.8573834215063333
MAPE for pred vs validation data:         0.5681625641578736
```



The parallel model does not help (much) with complex functions. Perhaps data re-uploading could be beneficial here?



Objective function is volatile. The parallel model degrades with the number of parameters (qubits x layers).



Quantum Fourier transforms Reflection

- Serial quantum Fourier transforms work amazingly well with a single qubit curve fitting
- With increased depth of a serial circuit, the performance decreases
- This is where the parallel quantum Fourier transform steps in and improves the outcome
- However, in both cases the more model parameters, the worse was the outcome (volatility of the objective function)
- Any deviation from a sine function, severely degrades the fit of both approaches
- The hypothesis that the parallel model could improve if we were to adopt the serial model's data re-uploading proved to be incorrect
- Worth noting that COBYLA and NELDER_MEAD optimisation excels in its task, L_BFGS_B is painfully slow, even though it provides good results (when finally completes)

Quantum neural networks

Pattern Matching

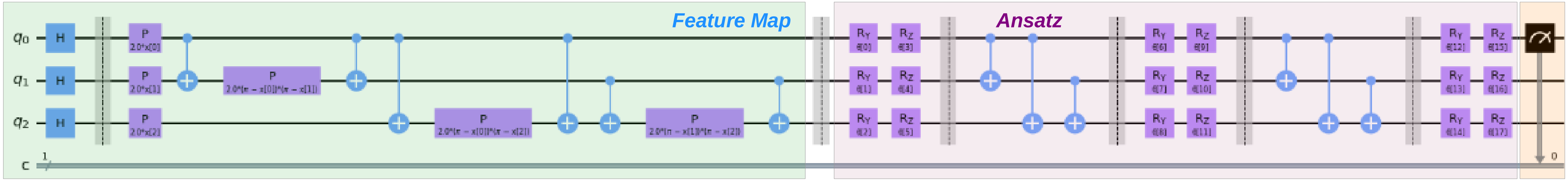
Abbas, Amira, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. "The Power of Quantum Neural Networks." Nature Computational Science 1, no. 6 (June 2021): 403–9. https://doi.org/10.1038/s43588-021-00084-1.

Schreiber, Amelie. "Quantum Neural Networks for FinTech." Medium, May 8, 2020. https://towardsdatascience.com/quantum-neural-networks-for-fintech-dddc6ac68dbf.

- A typical QNN consists of two main components, i.e. a feature map and an ansatz (also called variational model)
- The feature encodes the input data and prepares the quantum system state, using as many features as there are qubits
- The ansatz consists of several layers and, similarly to a classical NN, is responsible for inter-linking the layers - this is accomplished by trainable Pauli rotation gates and entanglement blocks
- Finally, the qubit states are measured and interpreted as QNN output

- In contrast to function / data fitting, QNNs are able to perform pattern matching, i.e. work with a sequence of values themselves rather than with the mapping between an index and values
- In the following experiments, we will adopt a sliding window approach to structuring the time series
- However, the standard QNN model does not lean itself to time series analysis, i.e.
  - You are limited to the TS window of size equal to the number of qubits

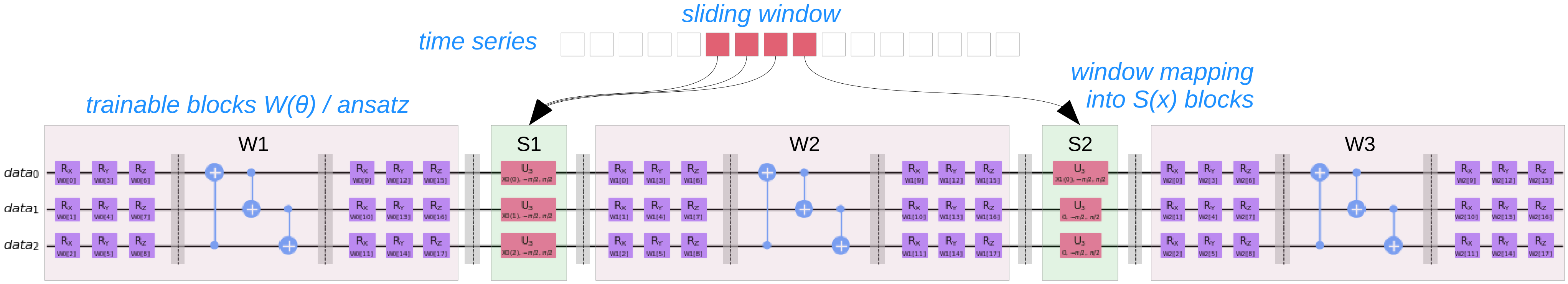
In Qiskit VQR Model



Quantum neural networks
Sliding windows / Serial model

- Experiments show that typical QNN (VQR) do not perform well with time series data
- The solution is to extend the Fourier quantum model into the multi-qubit QNN
- This required creation of a custom quantum circuit, which consists of encoding blocks $S_n(x)$ and trainable ansatz blocks $W_n(\theta_n)$
- The Fourier parallel model simply replicated the Sn(x) blocks, which limited the TS window size to the number of qubits, and which was tested to perform quite poorly

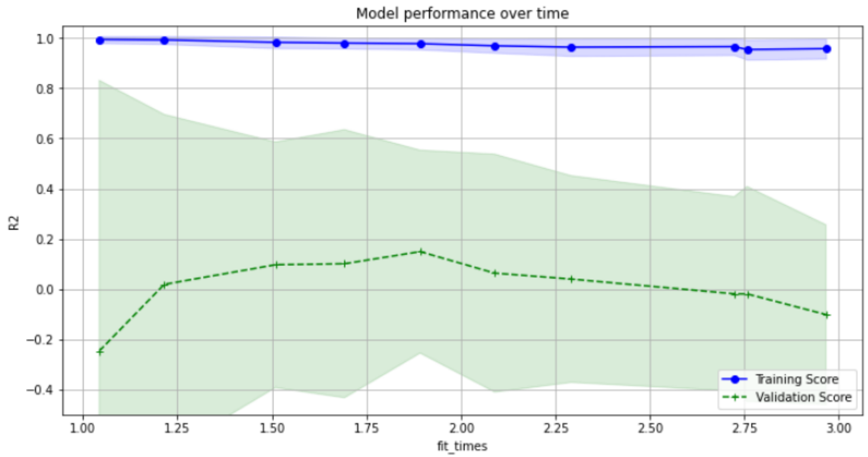
- An alternative was to adapt the Fourier serial model and distribute the TS window data across the encoding blocks $S_n(x_k)$, where each block would hold as many data points as there are qubits (k)
- Should the last block Sn(xk) be only partially filled with TS data, then the identity gates are used to make the complete block
- The circuit is then trained by optimising the parameters of trainable blocks $W_n(\theta_n)$

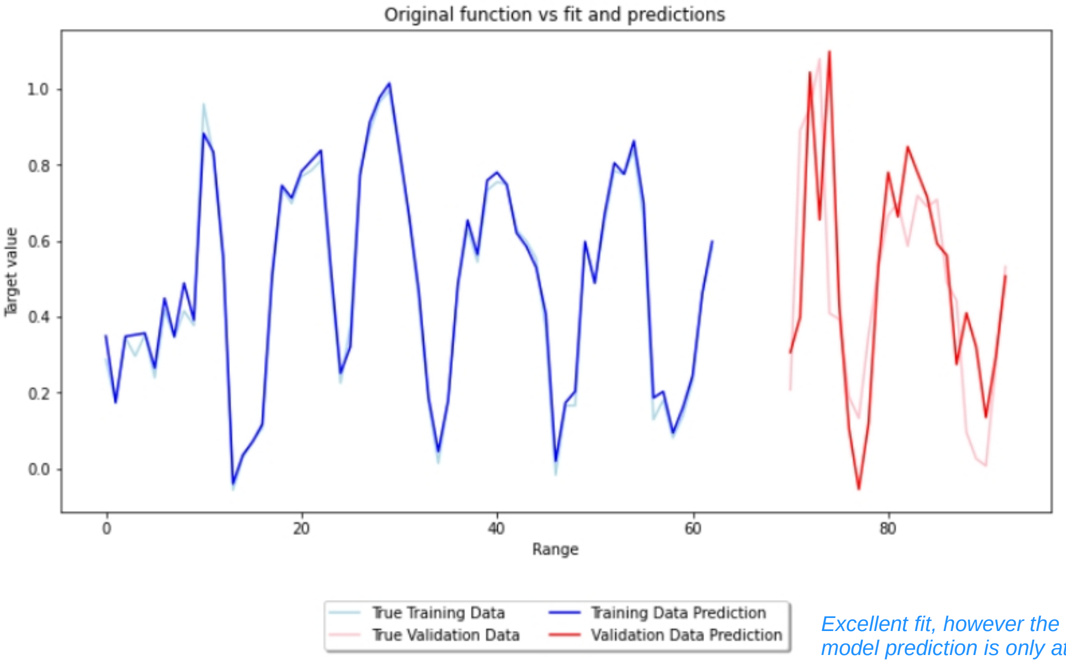


Experiment N1: Forecast for 2 Sins

Sliding Window
sklearn
MLPRegressor

2_sins(): samples_train=70, samples_valid=30
Data: wind=7, horizon=1
MLPRegressor: hidden_layer_sizes=(150,100,50), random_state=2022, max_iter=850, activation = 'relu', solver = 'adam', shuffle=True





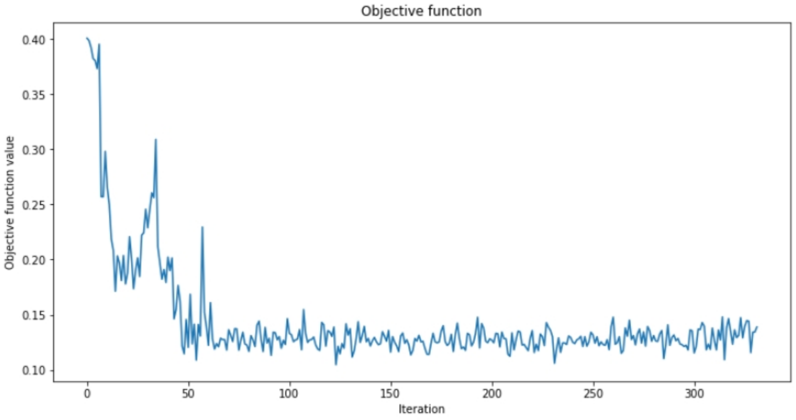

```
Training   score:        0.9904613870797965
Fake R2    score:        -4.642590797341893
MAPE for training:       0.1325727660135214

Validation score:        0.30134230110253646
Fake R2    score:        -11.184242847692133
MAPE for validation:     1.6791404176874565
```

Excellent fit, however the model prediction is only at R2=30% (possibly the model is overtrained)

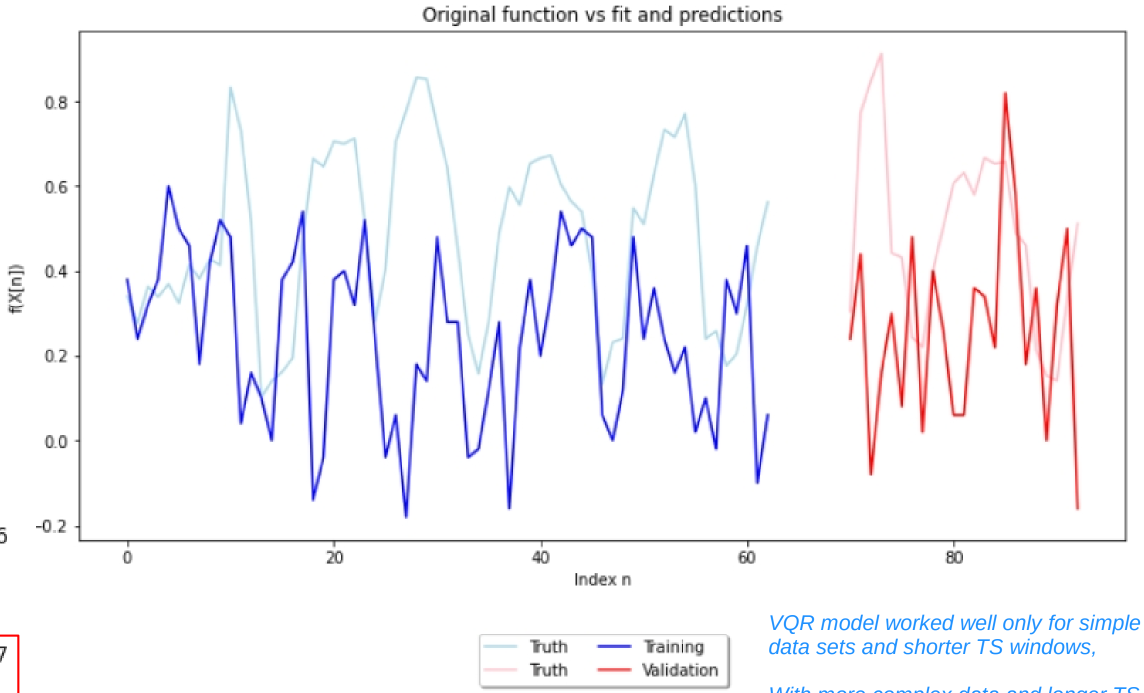
Experiment N2: Forecast for 2 Sins

Sliding Window VQR

sin(): samples_train=70, samples_valid=30
Data: wind=7, horizon=1 Prep: fmap=ZZFeatureMap(q=7, lays=1), ansatz=EfficientSU2(q=7, lays=2, ent="full", su2_gates=['ry', 'rz'])
VQR: observable='ZZZZZZZ', COBYLA()



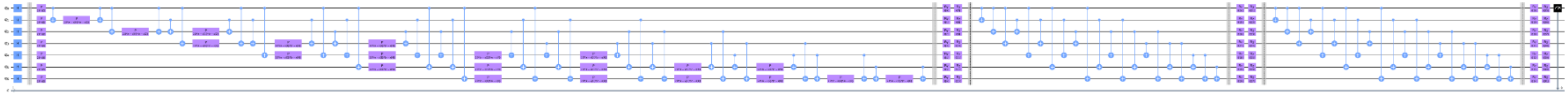

```
Minimum objective function value: (123, 0.10471365148566596)
```




```
R2 for pred vs training data:              -1.9726058878033546
MAPE for pred vs training data:             0.6699898313036804
Fake R2 for mean vs training data:         -9.116644938897453

R2 for pred vs validation data:            -2.0658331333082987
MAPE for pred vs validation data:           0.7104611881078806
Fake R2 for mean vs validation data:       -15.956157329019735
```

VQR model worked well only for simple data sets and shorter TS windows,

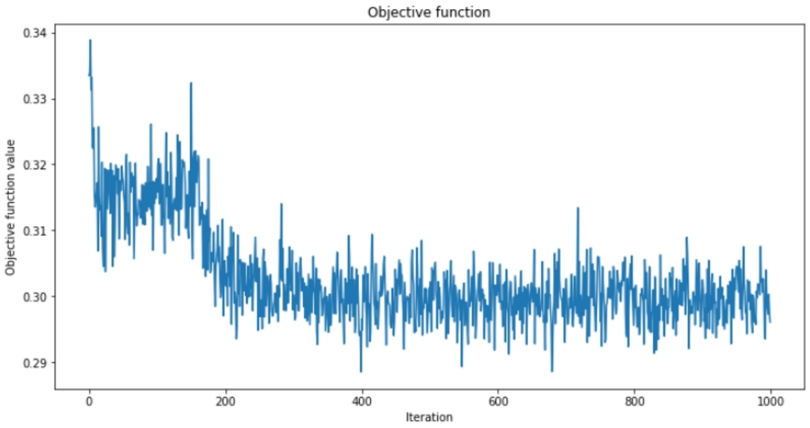
With more complex data and longer TS windows, its performance significantly degraded



Experiment N3: Forecast for 2 Sins

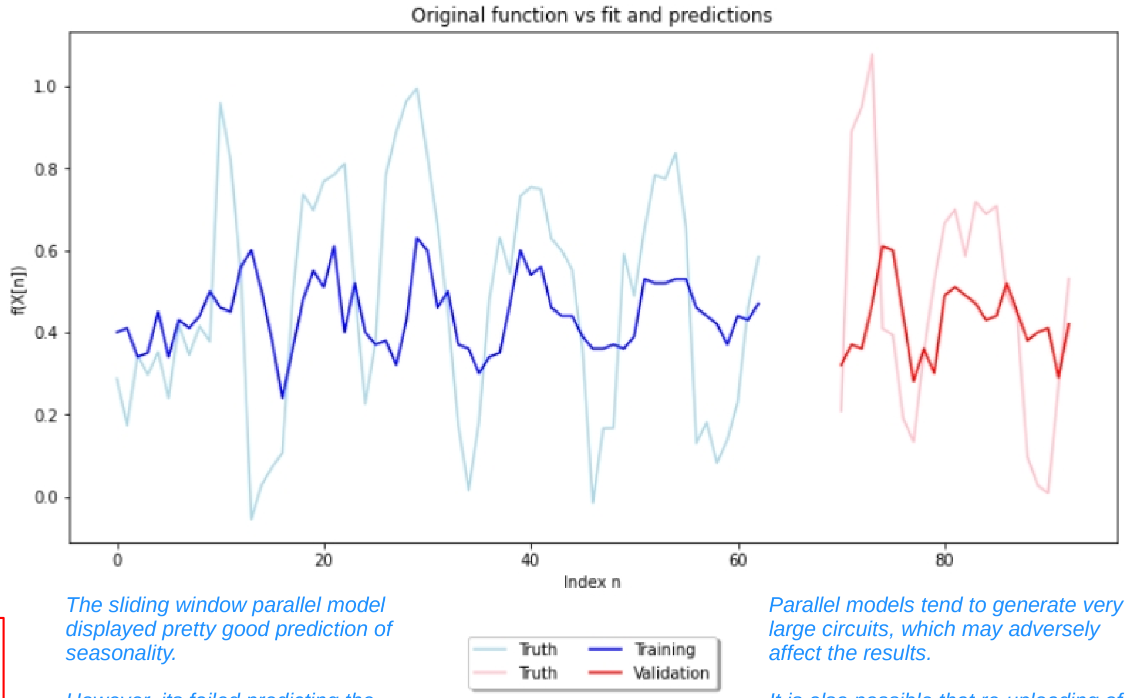
Sliding Window Parallel Model

2_sins(): samples_train=70, samples_valid=30
Data: wind=7, horizon=1
swindow_parallel_model+NNR+CircuitQNN: interpret=parity, qubits=7, ans_layers=3, optimizer=COBYLA()




```
Minimum objective function value: (399, 0.28848159138111823)
```

```
R2 for pred vs training data:              0.22890923678220454
Fake R2 for mean vs training data:         -0.44818249358929796
MAPE for pred vs training data:            1.7365546202086017

R2 for pred vs validation data:            0.022659435706280873
Fake R2 for mean vs validation data:       -0.33129136935703474
MAPE for pred vs validation data:          3.4162479029701087
```

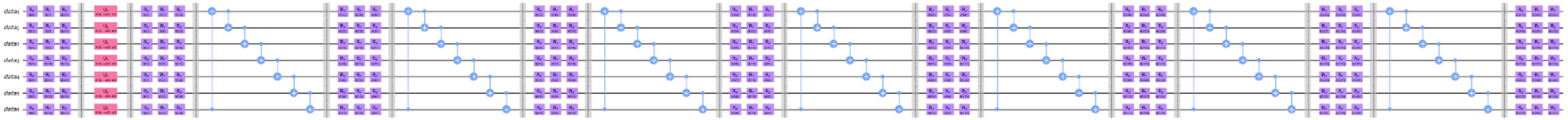


The sliding window parallel model displayed pretty good prediction of seasonality.

However, its failed predicting the data amplitude.

Parallel models tend to generate very large circuits, which may adversely affect the results.

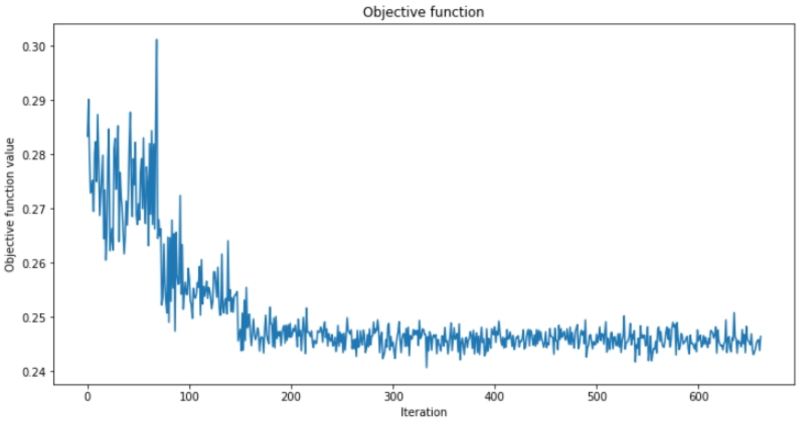
It is also possible that re-uploading of data may benefit the outcome!



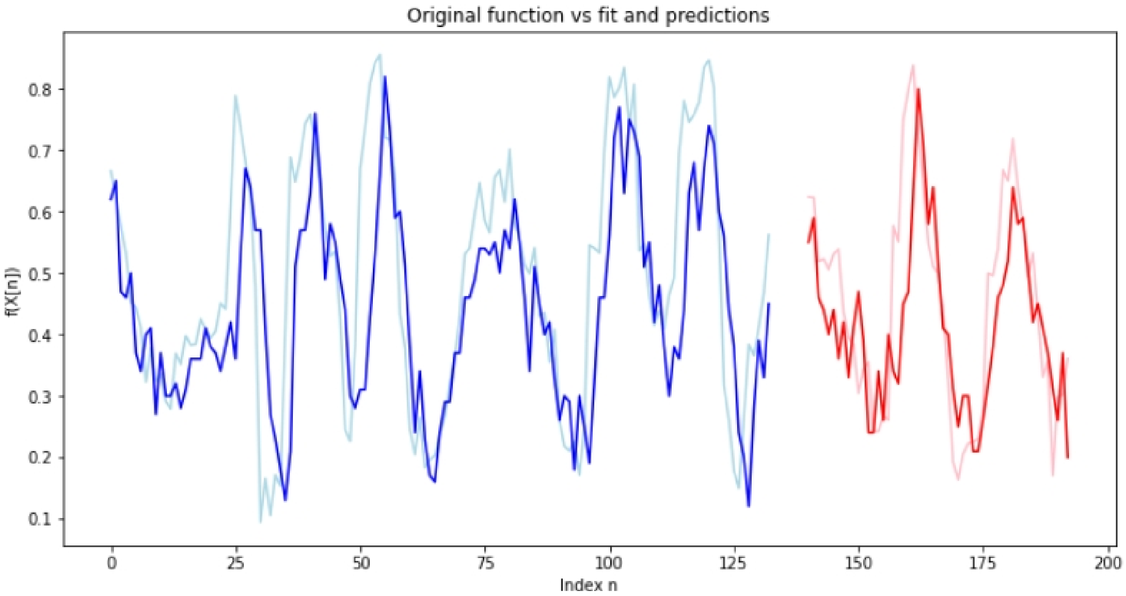
Experiment N7: Forecast for 2 Sins

Sliding Window Serial Model

2_sins(): samples_train=70, samples_valid=30
Data: wind=7, horizon=1
swindow_serial_model + NNR+CircuitQNN: interpret=parity, qubits=3, layers=3, optimizer=COBYLA()




```
Minimum objective function value: (333, 0.24063370363234937)
```



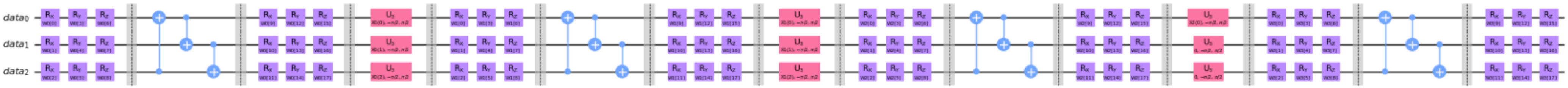

```
R2 for pred vs training data:              0.4216243799816321
Fake R2 for mean vs training data:         -0.15692687936155725
MAPE for pred vs training data:            0.2766545159136265

R2 for pred vs validation data:            0.594852151990251
Fake R2 for mean vs validation data:       -0.18588781421017253
MAPE for pred vs validation data:          0.2256624176380796
```

The sliding window serial model demonstrated good prediction of both seasonality and the signal amplitude.

The model does not re-upload data, however, it overloads TS data points.

This model can handle TS windows longer than the number of qubits available!

The model prediction significantly improved, well above the classical MLP!



Quantum neural networks Reflection

- QNNs are a promising approach to QTS forecasting.
- Unlike other QTS methods, QNNs are capable not only of data fitting but also pattern matching and prediction.
- However, the standard QNN model consisting of a feature map and a trainable ansatz, demonstrate poor performance when trained with more complex data.
- The proposed model for QTS forecasting, extends the single-qubit quantum Fourier serial model to work in a multi-qubit settings and more complex data.
- The model relies on the TS data continuity to reduce the need for re-uploading its input data, and instead overloads the qubits with blocks encoding the entire TS window of data points.
- The proposed model not only is able to encode more data than the number of its qubits, but the preliminary experiments also demonstrated the performance exceeding that of the classical MLP.

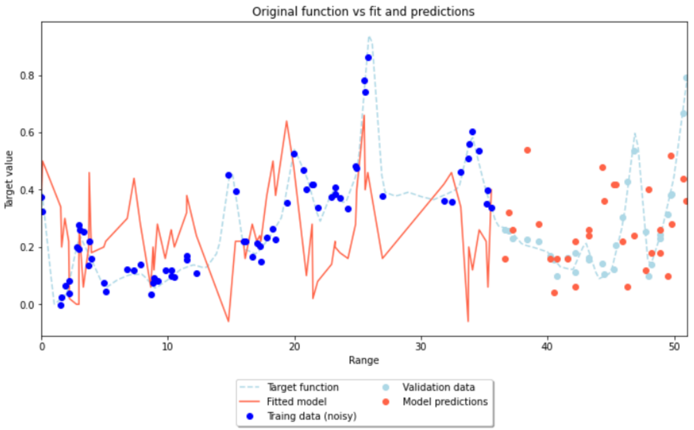
Other researchers in QNNs proposed quantum models of RNNs and LSTM!

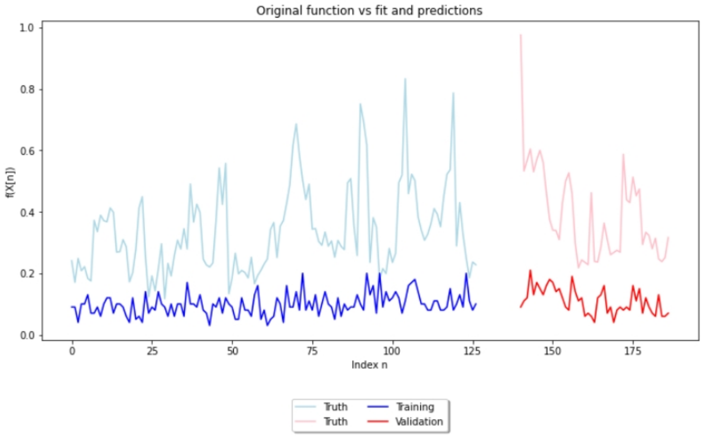
Bausch, Johannes. “Recurrent Quantum Neural Networks.” *Advances in Neural Information Processing Systems* 33 (2020): 1368–79.

Chen, Samuel Yen-Chi, Shinjae Yoo, and Yao-Lung L. Fang. “Quantum Long Short-Term Memory.” In ICASSP 2022-2022 *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8622–26. IEEE, 2022.

Working with world data

- Our QTSA research features real-world applications. For example, our work involved records of beer sales in USA.
- The preliminary experiments indicate that more work is required to make the proposed model practical.
- The future work will include:
  - Adoption of different circuit measurement strategies and interpretation of results.
  - Inclusion of non-linearities to mimic NN activation functions.
  - Dealing with larger TS horizons and non-stationary TS
  - Most importantly:
    Exploration of avenues for QTSA to demonstrate real quantum advantage, e.g. in TS anomaly detection, adoption of stochastic TS analysis, etc.





Bird-view of Quantum Time Series Analysis
Summary, reflections and questions

TS processing requires data storage

Variational quantum regression is too simplistic

QNNs with data re-uploading and overloading are key

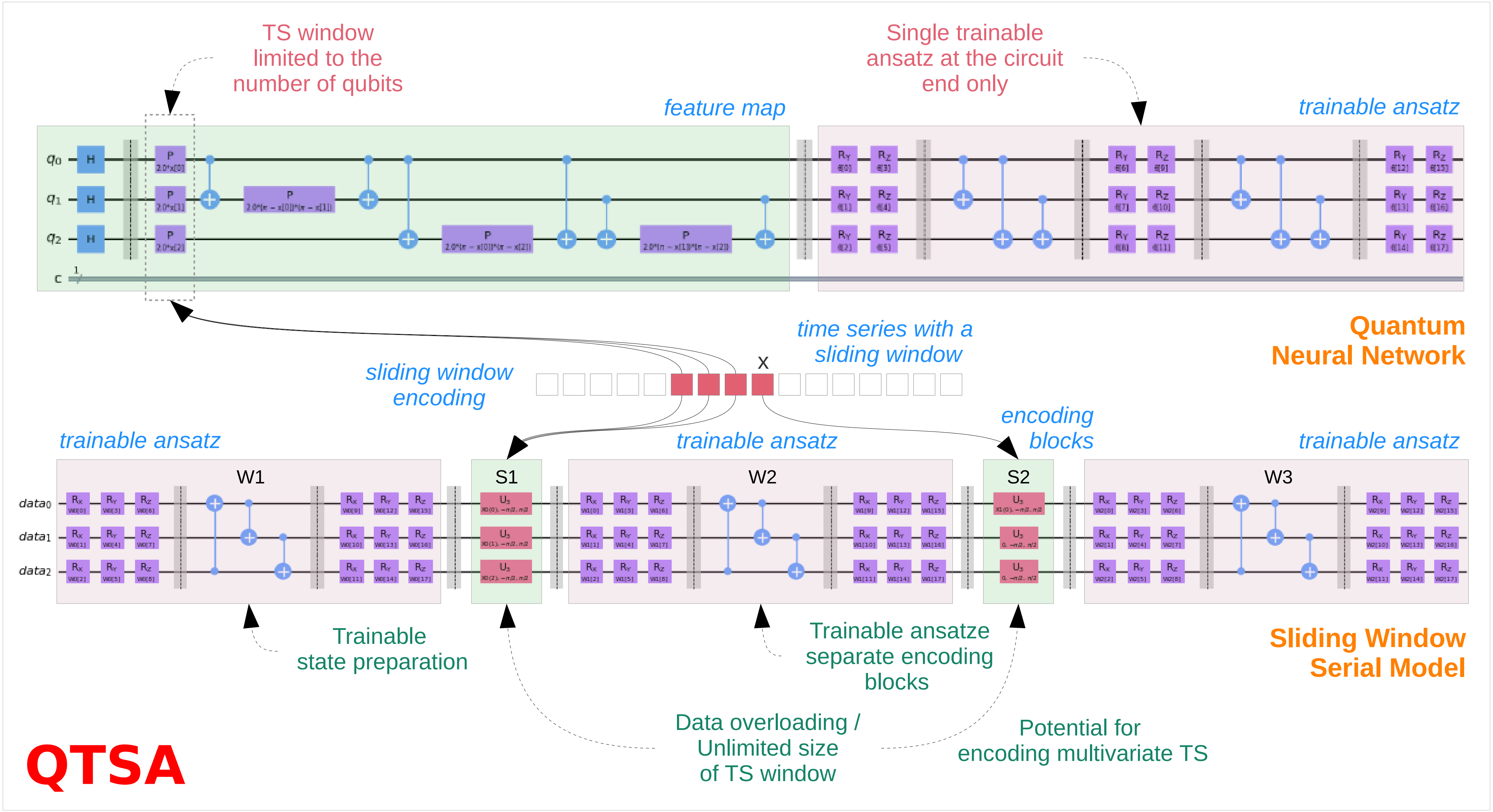
Quantum systems have no memory

Quantum Fourier transforms are promising

Variational quantum models effectively simulate memory

QC creates opportunities for TSA

Quantum neural nets suggest the solution to QTSA



TS window
limited to the
number of qubits
Single trainable
ansatz at the circuit
end only
feature map
trainable ansatz
Quantum
Neural Network
time series with a
sliding window
X
sliding window
encoding
encoding
blocks
trainable ansatz
trainable ansatz
trainable ansatz
W1
S1
W2
S2
W3
Trainable
state preparation
Trainable ansatze
separate encoding
blocks
Sliding Window
Serial Model
Data overloading /
Unlimited size
of TS window
Potential for
encoding multivariate TS
QTSA