The aims of this session:

To explore and understand various design issues while developing a complex quantum autoencoder

Development of Quantum Autoencoders

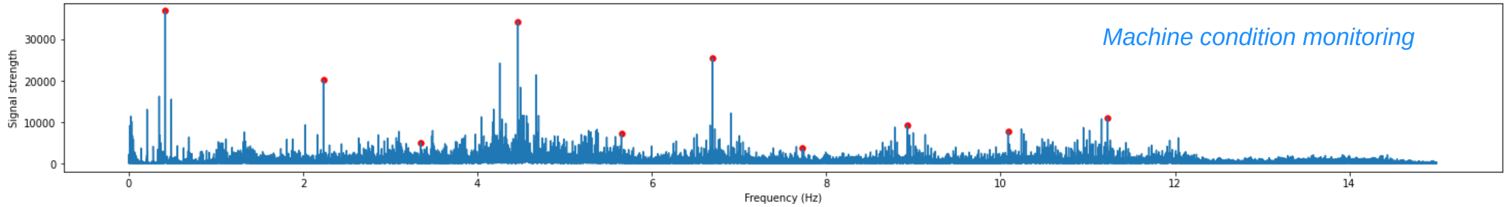
Worst case scenario: denoising time-series and signals

Jacob L. Cybulski
Enquanted, Melbourne, Australia

Sebastian Zając
SGH Warsaw School of Economics, Warsaw, Poland

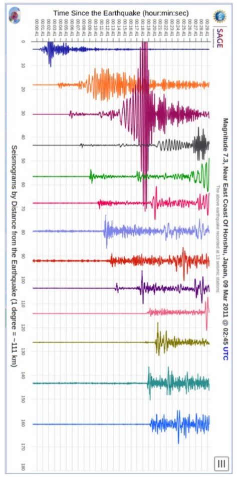
5 October 2024, Washington DC Quantum Computing Meetup

Problem



Machine condition monitoring

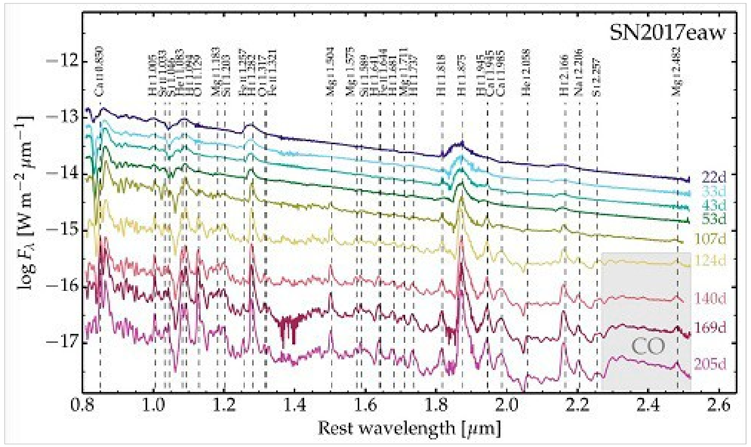
EEG



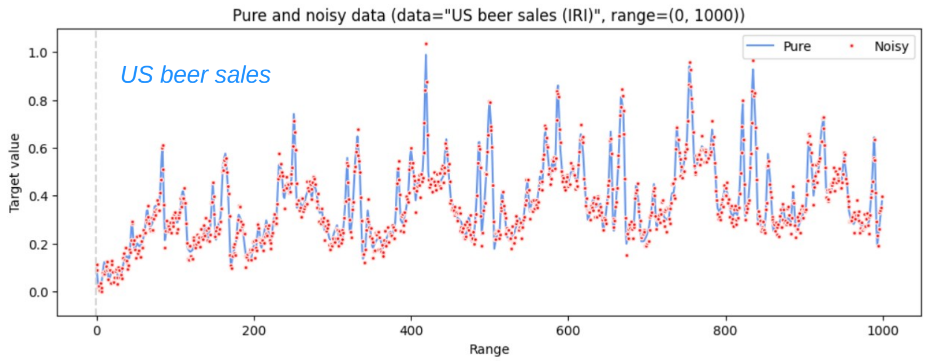
Seismogram

Can quantum machine learning assist in detection of complex patterns, such as anomalies, in time series and signals, from the preceding sequences of data.

Sample applications include: machine condition monitoring, astronomical observations, nationwide marketing and sales, earthquake prediction, EEG or ECG analysis.



Astronomical observation



US beer sales

Acknowledgements:

Gearbox and Vibration Analysis ML, 2023, Nakkeeran, Kaggle.com

Gemini/GNIRS spectra, 2017, NOIRLab, Wikimedia.

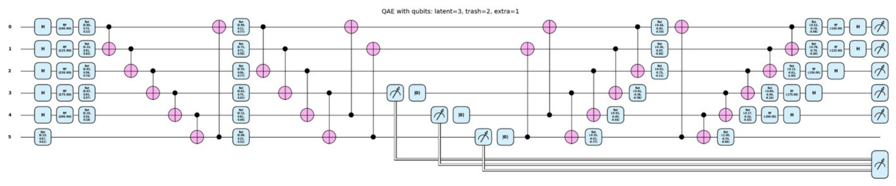
Bronnenberg, B.J., Kruger, M.W., Mela, C.F., 2008. Database Paper —The IRI Marketing Data Set. Marketing Science 27, 745–748. https://doi.org/10.1287/mksc.1080.0450

Earthquake, Mag 7.3, East Coast of Honshu, Japan, 2011, The Global Seismogram Viewer, http://ds.iris.edu/gsv

EEG of brain and heart action, 2012, Otoomuch, Wikimedia.

Presenter

Jacob Cybulski
quantum@jacobcybulski.com

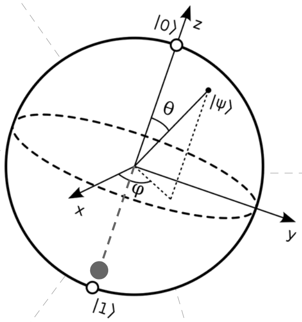


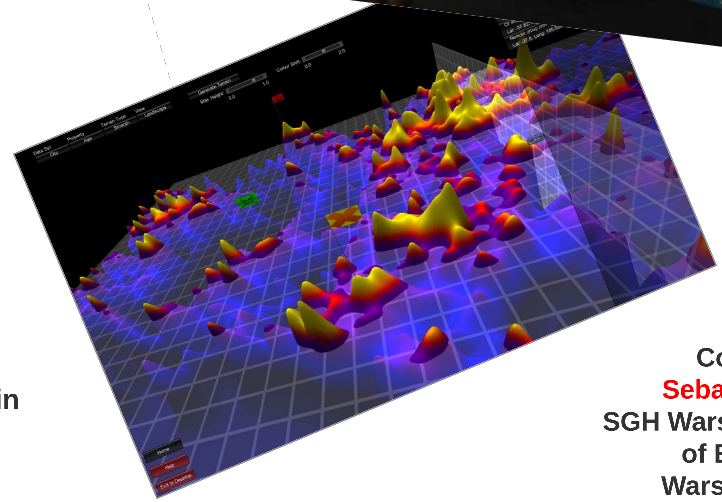
Founder
Researcher
Consultant
Author
at Enquanted

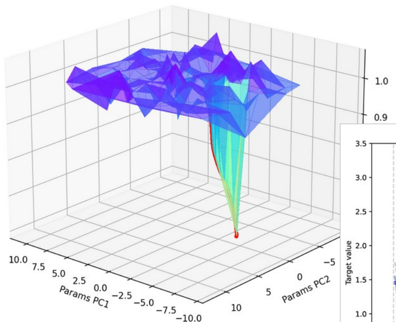
Melbourne
Australia

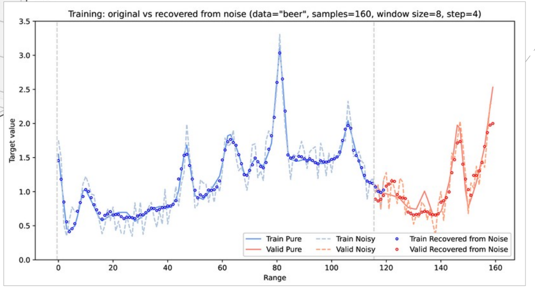
Projects
- Quantum computing
- Quantum machine learning
- Quantum time series analysis and anomaly detection
- Classical machine learning
- Data visualisation

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









Research collaboration and supervision of research students in QC + QML

Collaborator:
Sebastian Zając
SGH Warsaw School of Economics, Warsaw, Poland

Quantum Autoencoder (for Time Series)

Autoencoders (AE) are ML models able to to compress input into its essential features and then recover the original info

They lose the infrequent or insignificant info, such as anomalies and noise

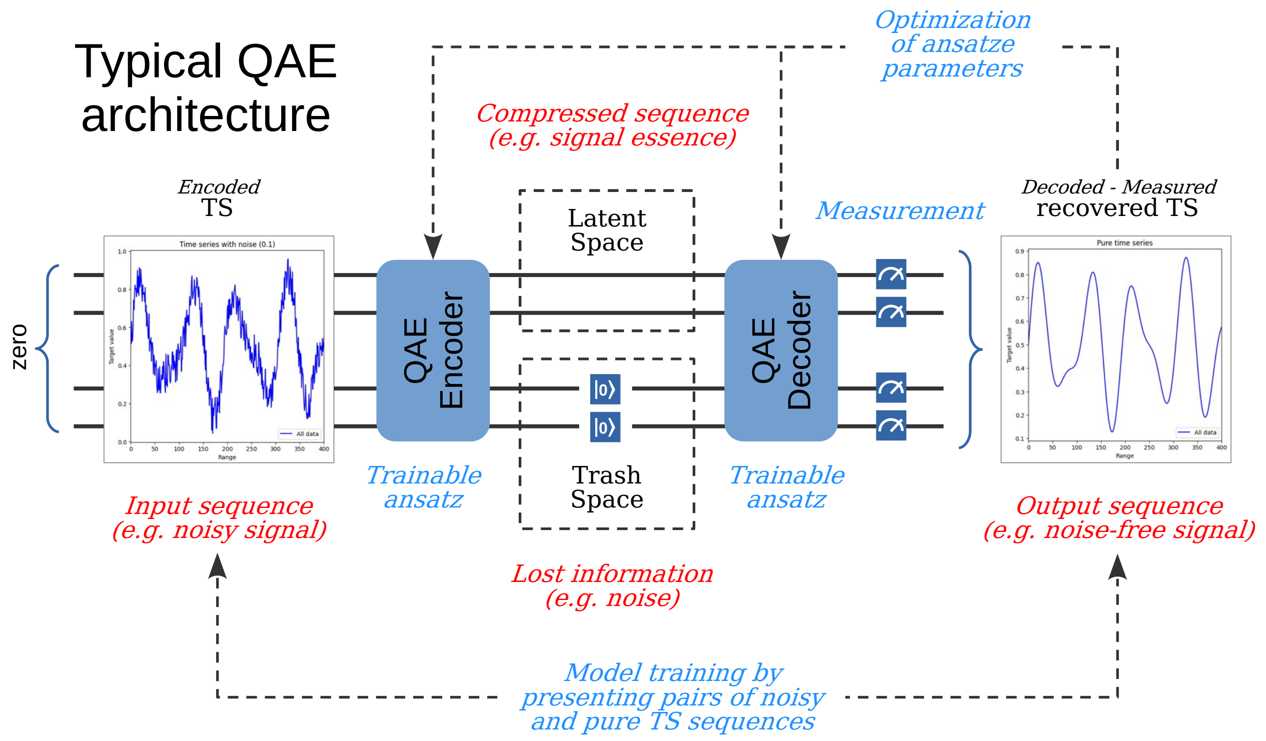
Used for data denoising and anomaly detection, e.g. in images and signals

There are few applications of QML methods to time-series analysis, QAE even fewer

QAEs have the potential to deal with complex noise and anomaly patterns

Training of QAEs is difficult, due to:

- Potentially many features (e.g. TSs) (lots of qubits and/or parameters)
- Complex measurement strategies
- Unsupervised learning (we do not know what is noise)
- Possibility of barren plateaus



Typical QAE architecture
Optimization of ansatze parameters
Compressed sequence (e.g. signal essence)
Encoded TS
Latent Space
Measurement
Decoded - Measured recovered TS
zero
Time series with noise (0.1)
Pure time series
QAE Encoder
QAE Decoder
|0⟩
|0⟩
Trash Space
Trainable ansatz
Trainable ansatz
Input sequence (e.g. noisy signal)
Output sequence (e.g. noise-free signal)
Lost information (e.g. noise)
Model training by presenting pairs of noisy and pure TS sequences

In QAE development, the key concerns include: *overall model architecture, data encoding and decoding, ansatz design and its parameters optimisation strategy*

Input encoding / embedding

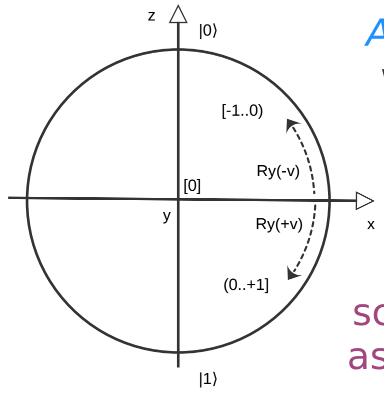
for QAE processing

In general, QAE input and output is an unrestricted collection of real values (floats) – this guided our selection of data encoding methods.

We rejected the following encoding methods:

Basis encoding, with qubits acting as bits in the encoded number (logical / int) to be processed later in the circuit.

QRAM encoding, where all possible inputs are known in advance, pre-coded in a circuit, and used by reference.

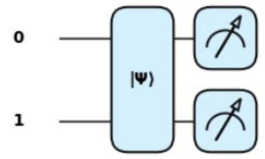


Angle encoding suits QAE design, with input values represented as qubit state rotations (float).

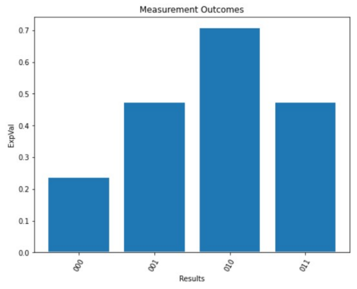
In our experiments we used angle encoding relative to |+⟩ state, with values ∊ [-1, 1] scaled to arange [0, π], and coded as rotations up (<0) or down (≥0).

Amplitude encoding is probably the least understood, however, it is one of the most useful encoding schemes - attractive for QAEs.

It embeds input as a circuit state normally measured on output, i.e. each data point is encoded as expectation value of *multi-qubit measurement (int / float)*.



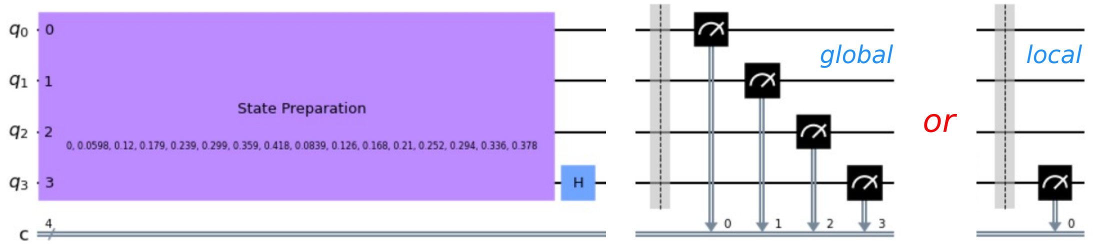
The problem with this encoding scheme is that for each unique input value, the structure of encoding gates is different. The circuit is not differentiable, which may be suitable for simulators, but difficult to use with GPUs and QPUs.



Example: data encoded as ψ was normalised vector [1/8, 2/8, 3/8, 2/8]. The measurement reflects the input data proportions.

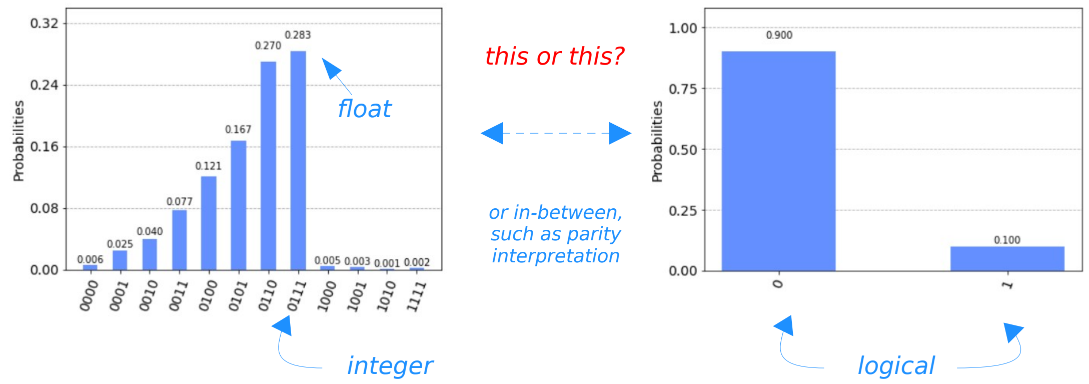
Maria Schuld and Francesco Petruccione. *Machine Learning with Quantum Computers. 2nd ed*. Springer, 2021. http://link.springer.com/book/10.1007/978-3-030-83098-4.

Measurement & Interpretation



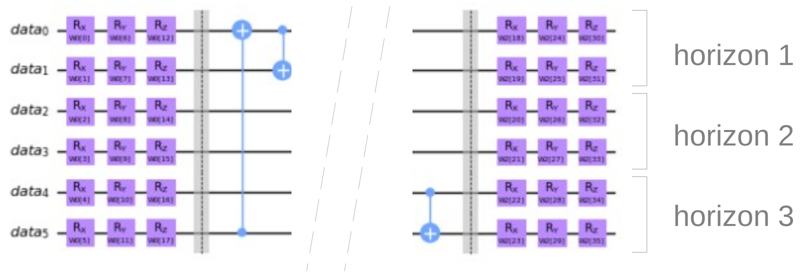
There are many ways of decoding the circuit state to form classical output data, e.g. we can:

- *measure all qubits*
  (as related to the global cost function)
- *measure a selection of qubits*
  (as related to the local cost function)
- *measure the circuit state in different ways*
  (e.g. as counts, expvals or probabilities)
- *reinterpret circuit measurements*
  into different combinations of outcomes, e.g. to predict larger TS horizons (future)



Repeated circuit measurement can be interpreted as outcomes of different numeric types, e.g. as a:

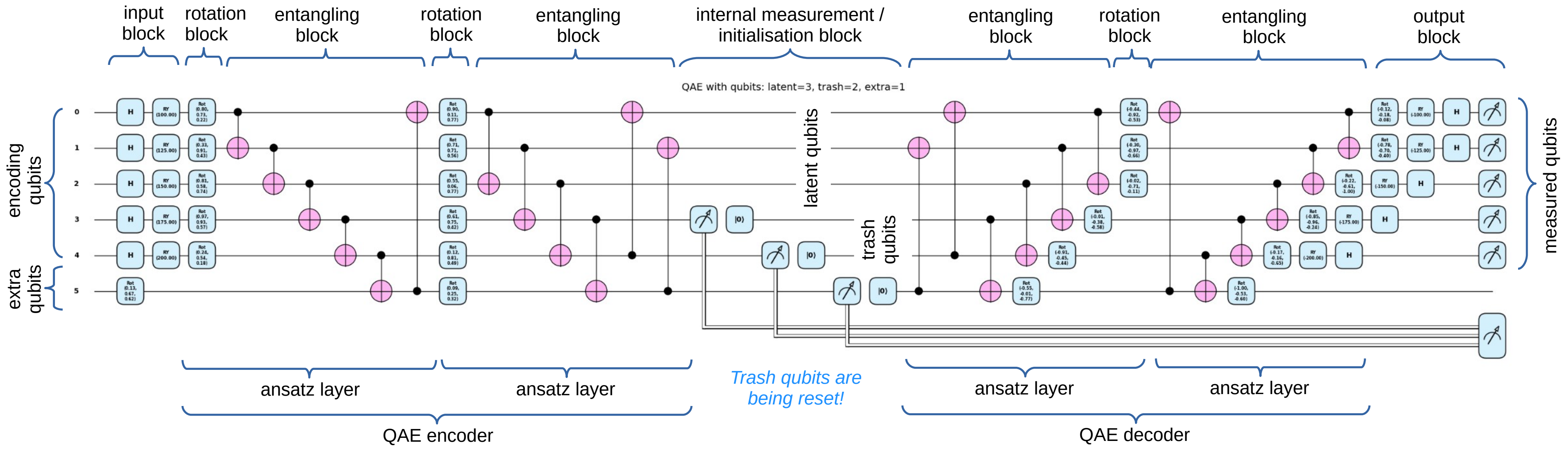
- *binary outcome*
  (e.g. a single qubit measurement),
- *bitwise representation of an integer number*
  (e.g. most frequent combination of multi-qubit measurements), or
- *value of a continuous variable*
  (e.g. expectation value of a specific outcome).



Anatomy of QAE Ansatze

QAE encoder and decoder (Qiskit)

This model features mid-circuit measurement which is not a unitary operation, hence it is not differentiable and hard to optimise.



QAE encoder and decoder are often symmetric (as shown here)

They are parametrized circuits (ansatze), arranged into layers of trainable rotation and entangling blocks

Ansatze may be of a different size than the requirements of input/output blocks

The selection of the optimizer of ansatze parameters requires some preliminary investigation of their effectiveness

This depends on the model architecture, ansatz design, data encoding and decoding, as well as the nature of training data

In our project we evaluated gradient based optimizers (ADAM and SPSA) as well as linear and non-linear approximation methods (such as COBYLA and BFGS) – COBYLA was adopted

Experiments
with QAE denoising TSs

Subsequently, a series of over 60 (Qiskit) experiments were conducted to determine the optimum QAE model characteristics

First, it was required to determine the time-series window size, which implied the size of QAE model input and output blocks

Then three circuit parameters were varied, i.e. the size of latent (and trash) space, the number of additional qubits required by the ansatze, and the type and the number of rotational (y and xy) / entangling layers

The optimum model parameters were selected based on the model validation scores (MAE)

The best QAE model's performance was comparable to the best equivalent deep learning model (14 additional experiments with PyTorch models)

Qiskit state vector simulator
Best quantum model (7, 3, 2)
Number of parameters: 180
Number of iterations: 2,000
Speed of model training: 15 mins

PyTorch
Equivalent classical DL model (7)
Number of parameters: 9,741
Number of iterations: 30,000
Speed of model training: 20 secs

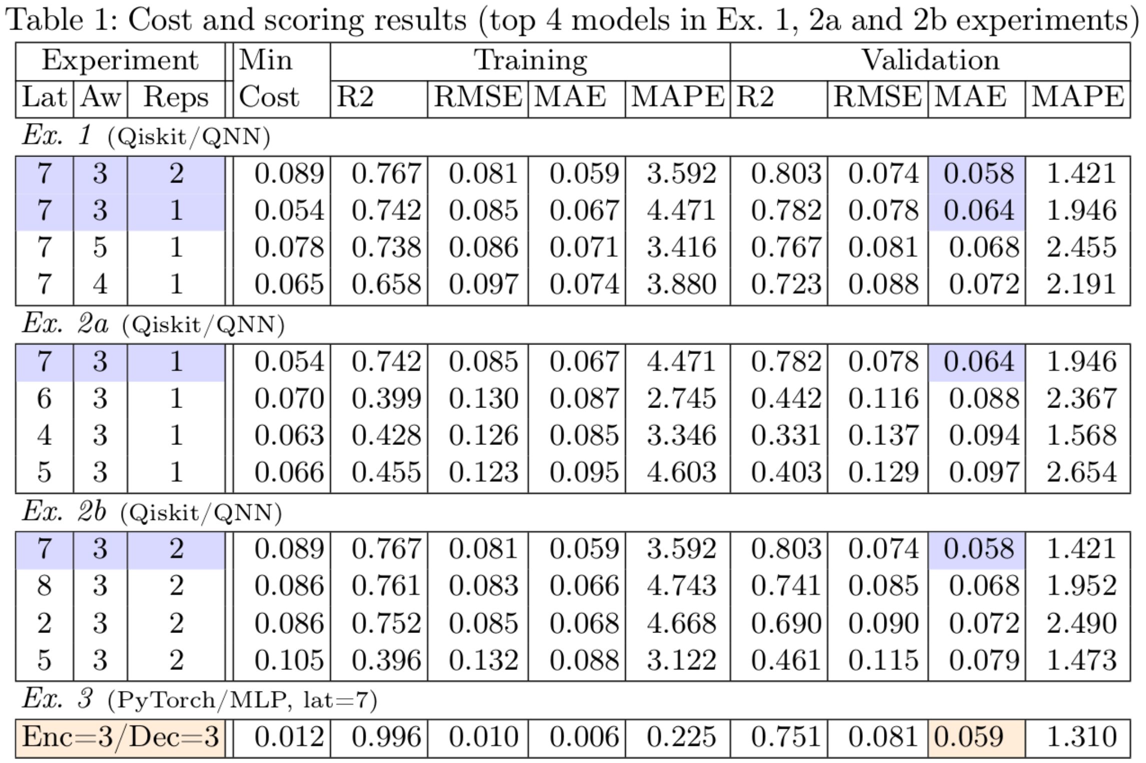
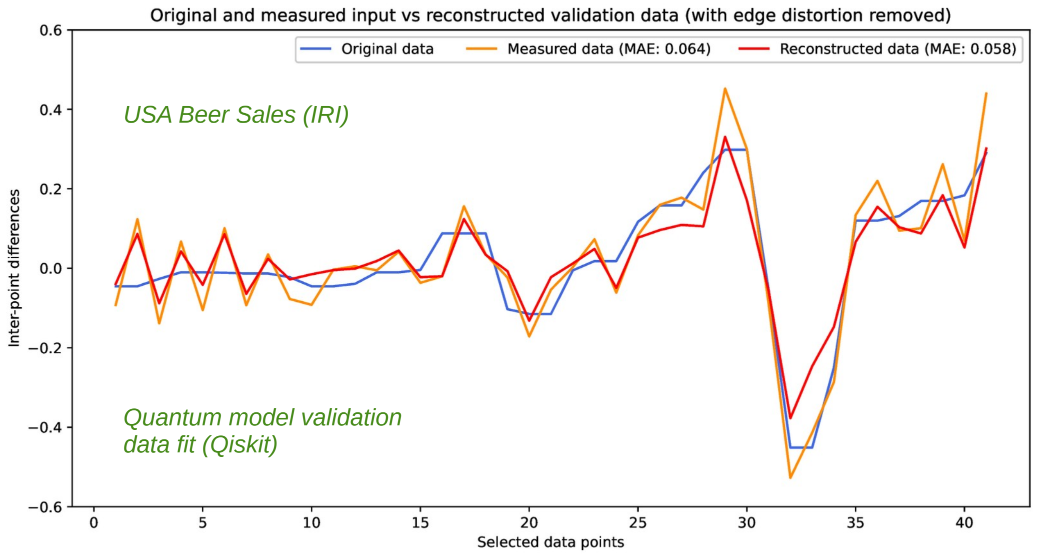


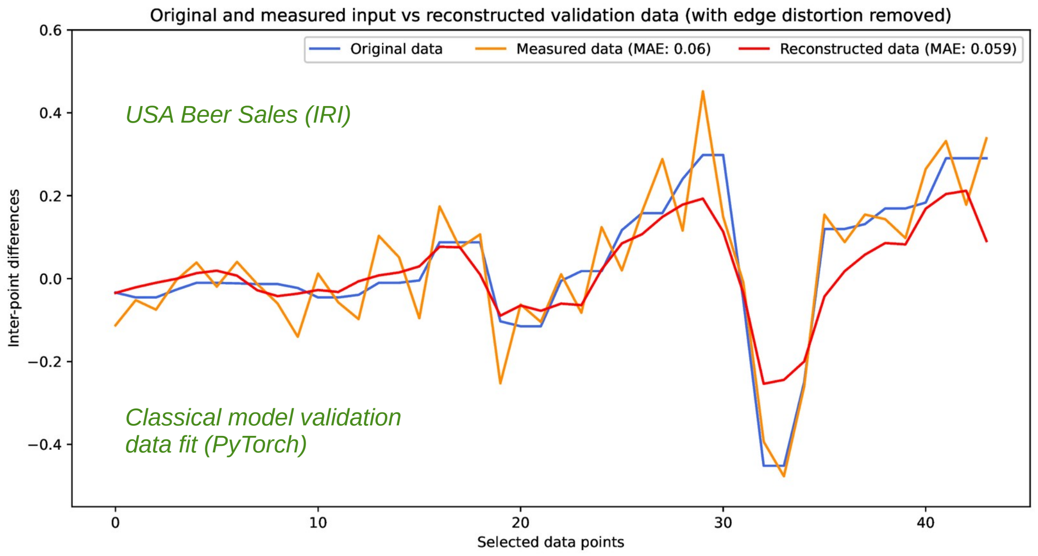
Table 1: Cost and scoring results (top 4 models in Ex. 1, 2a and 2b experiments)

Experiment			Min	Training				Validation			
Lat	Aw	Reps	Cost	R2	RMSE	MAE	MAPE	R2	RMSE	MAE	MAPE
Ex. 1 (Qiskit/QNN)											
7	3	2	0.089	0.767	0.081	0.059	3.592	0.803	0.074	0.058	1.421
7	3	1	0.054	0.742	0.085	0.067	4.471	0.782	0.078	0.064	1.946
7	5	1	0.078	0.738	0.086	0.071	3.416	0.767	0.081	0.068	2.455
7	4	1	0.065	0.658	0.097	0.074	3.880	0.723	0.088	0.072	2.191
Ex. 2a (Qiskit/QNN)											
7	3	1	0.054	0.742	0.085	0.067	4.471	0.782	0.078	0.064	1.946
6	3	1	0.070	0.399	0.130	0.087	2.745	0.442	0.116	0.088	2.367
4	3	1	0.063	0.428	0.126	0.085	3.346	0.331	0.137	0.094	1.568
5	3	1	0.066	0.455	0.123	0.095	4.603	0.403	0.129	0.097	2.654
Ex. 2b (Qiskit/QNN)											
7	3	2	0.089	0.767	0.081	0.059	3.592	0.803	0.074	0.058	1.421
8	3	2	0.086	0.761	0.083	0.066	4.743	0.741	0.085	0.068	1.952
2	3	2	0.086	0.752	0.085	0.068	4.668	0.690	0.090	0.072	2.490
5	3	2	0.105	0.396	0.132	0.088	3.122	0.461	0.115	0.079	1.473
Ex. 3 (PyTorch/MLP, lat=7)											
Enc=3/Dec=3			0.012	0.996	0.010	0.006	0.225	0.751	0.081	0.059	1.310

Noise removal
(Qiskit)

The best QAE model's ability to remove noise from signals (left) was comparable to (but not better than) the best equivalent DL model (right).





In training, QAE models seem to learn (red) by reducing the amplitude between noisy signals (orange) and pure signals (blue), while retaining the shape of noise.

Classical CAE models seem to be better at fitting the intended signals shape (red), while reducing the amplitude between noisy signals (orange) and pure signals (blue).

On metrics, CAEs excelled in models training performance.
However, in validation CAEs and QAEs performed at the same level.

What problems were discovered

Dealing with the complexity of the parameter space

Solution:
PennyLane/
PyTorch
approach to QAE
development

An approach adopted in the QAE creation was to rely on the VQA development in Qiskit

One of the issues found to affect the QAE training performance was:

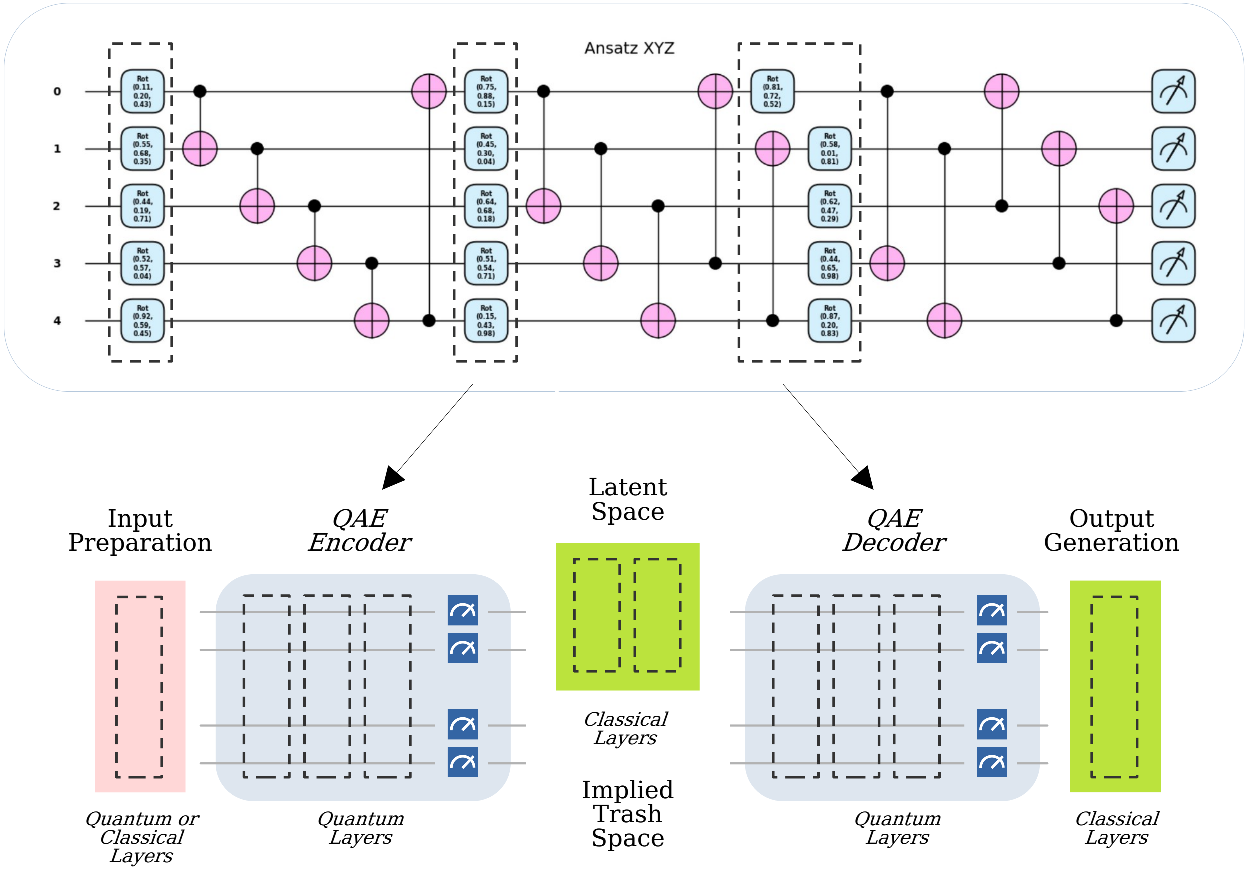
Dealing with deep quantum circuits consisting of large numbers of unstructured parameters

The currently pursued solution is to explore PennyLane / PyTorch ability to create hybrid models of well integrated quantum and classical components.

Large quantum models can be decomposed into classical DL NNs and a number of smaller quantum circuits.

Their parameters can be structured into layers so that they could be managed effectively by PyTorch during the optimisation process.

Qiskit recently adopted a similar open source framework “torchquantum”.



PennyLane / PyTorch Neural Network
of classical and quantum components

Anatomy of QAE Ansatze

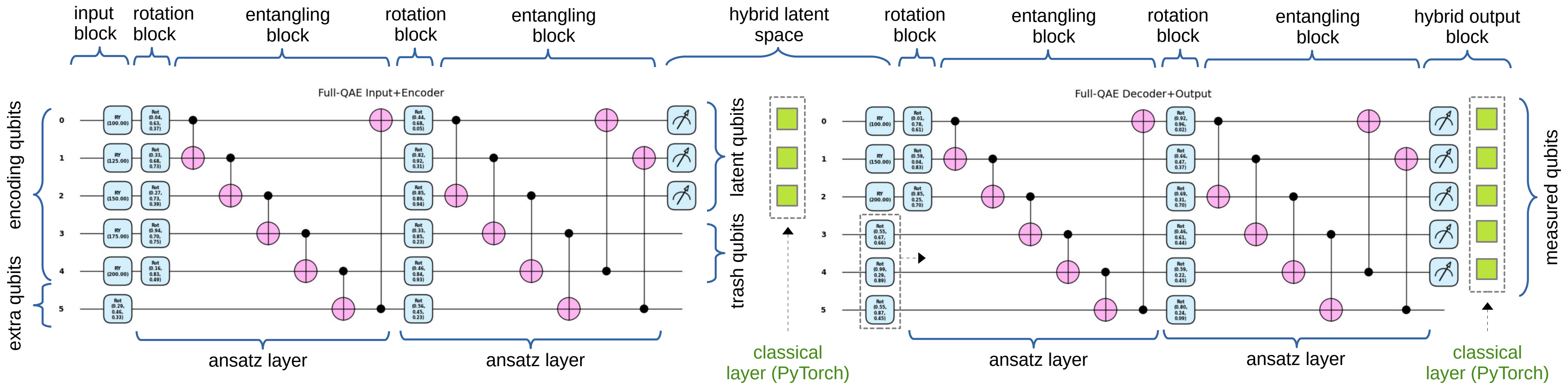
QAE encoder and decoder (PennyLane)

Note that classical layers are optional, however they greatly improve the model performance when running on a quantum simulator.

They can also add features not available in pure quantum models (e.g. nonlinearity).

They may, however, prevent quantum advantage.

It is a "minimum" hybrid model



QAE encoder and decoder do not need to be symmetric (here, they are not)

The hybrid QAE separated the encoder and decoder into two *shallow circuits*, which can be trained very effectively and fast.

However, hybrid QAEs lose some quantum information, to the detriment of their function.

PennyLane and PyTorch have excellent support for *gradient manipulation*, offering several highly efficient gradient optimisers.

Hence, we adopted a *Nadam optimiser*.

Note that Qiskit also provides some support for passing gradients into its optimisers, however, this is not being highlighted as Qiskit feature.

Comparing results
(PennyLane vs PyTorch)

USA beer sales (IRI)

The experiments show:

The larger the QAE latent space, the better learning
(the accepted idea that reducing latent space helps abstraction is wrong)

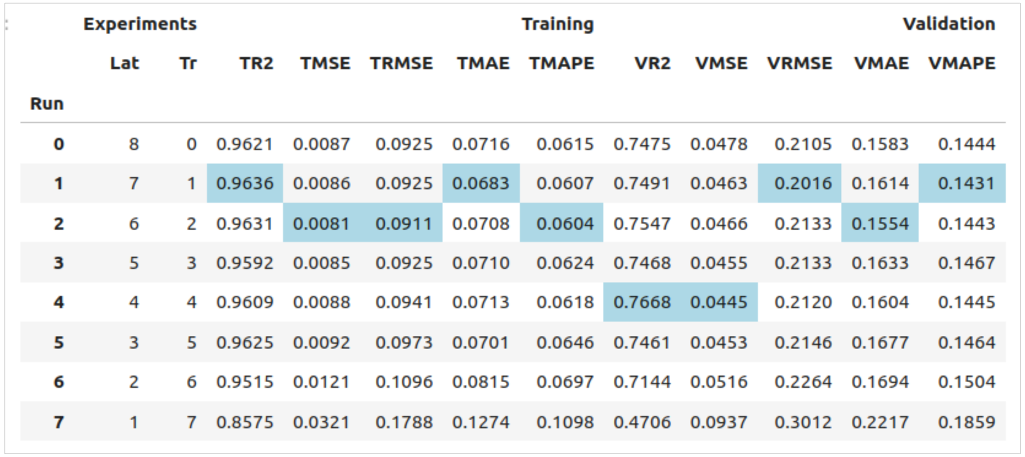
There is an optimum depth for the QAE model.

PennyLane “minimum” hybrid models outperformed Qiskit models in training, but not in validation.

Within the limit of 1000 epochs, QAE outperformed CAE.

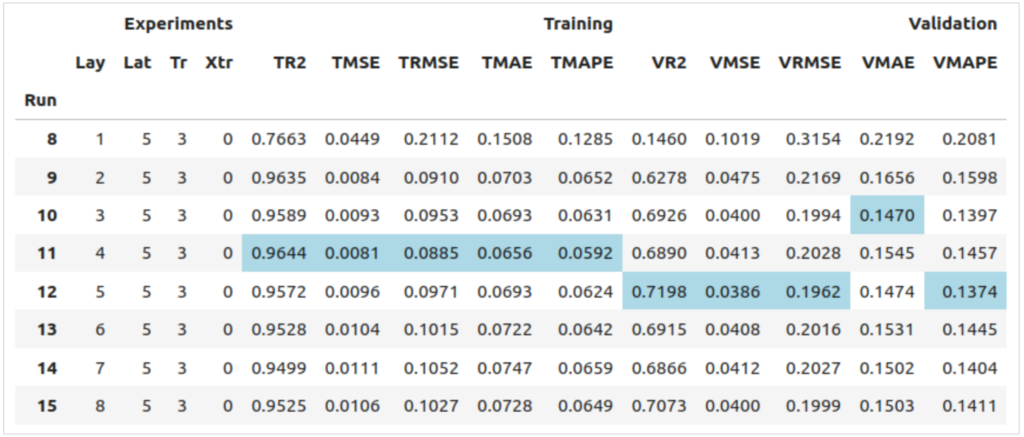
In general, QML models on simple tasks (such as DL AE) do not outperform the classical models – so to gain quantum advantage you need to pick the application very carefully.

Varying the latent space: DL CAE model in PyTorch @ 1000 epochs



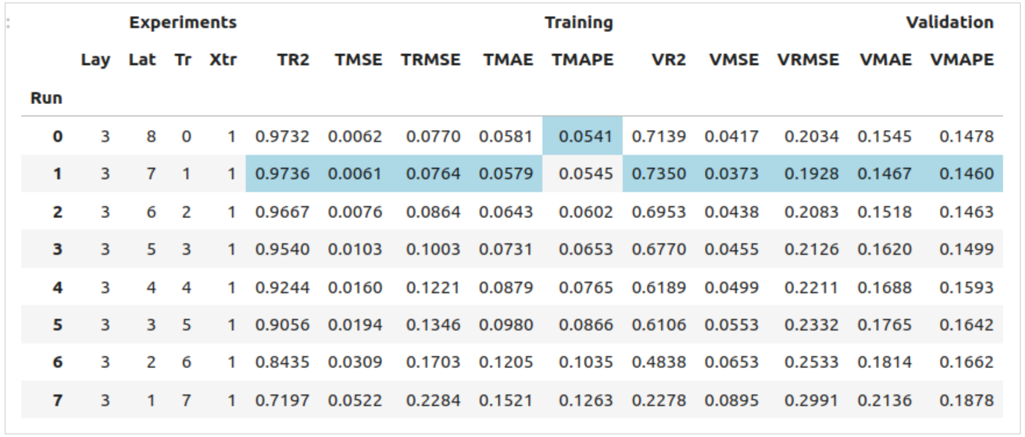
	Experiments		Training					Validation				
	Lat	Tr	TR2	TMSE	TRMSE	TMAE	TMAPE	VR2	VMSE	VRMSE	VMAE	VMAPE
Run												
0	8	0	0.9621	0.0087	0.0925	0.0716	0.0615	0.7475	0.0478	0.2105	0.1583	0.1444
1	7	1	0.9636	0.0086	0.0925	0.0683	0.0607	0.7491	0.0463	0.2016	0.1614	0.1431
2	6	2	0.9631	0.0081	0.0911	0.0708	0.0604	0.7547	0.0466	0.2133	0.1554	0.1443
3	5	3	0.9592	0.0085	0.0925	0.0710	0.0624	0.7468	0.0455	0.2133	0.1633	0.1467
4	4	4	0.9609	0.0088	0.0941	0.0713	0.0618	0.7668	0.0445	0.2120	0.1604	0.1445
5	3	5	0.9625	0.0092	0.0973	0.0701	0.0646	0.7461	0.0453	0.2146	0.1677	0.1464
6	2	6	0.9515	0.0121	0.1096	0.0815	0.0697	0.7144	0.0516	0.2264	0.1694	0.1504
7	1	7	0.8575	0.0321	0.1788	0.1274	0.1098	0.4706	0.0937	0.3012	0.2217	0.1859

Varying the circuit depth: quantum model in PennyLane + PyTorch @ 1000 epochs



	Experiments				Training					Validation				
	Lay	Lat	Tr	Xtr	TR2	TMSE	TRMSE	TMAE	TMAPE	VR2	VMSE	VRMSE	VMAE	VMAPE
Run														
8	1	5	3	0	0.7663	0.0449	0.2112	0.1508	0.1285	0.1460	0.1019	0.3154	0.2192	0.2081
9	2	5	3	0	0.9635	0.0084	0.0910	0.0703	0.0652	0.6278	0.0475	0.2169	0.1656	0.1598
10	3	5	3	0	0.9589	0.0093	0.0953	0.0693	0.0631	0.6926	0.0400	0.1994	0.1470	0.1397
11	4	5	3	0	0.9644	0.0081	0.0885	0.0656	0.0592	0.6890	0.0413	0.2028	0.1545	0.1457
12	5	5	3	0	0.9572	0.0096	0.0971	0.0693	0.0624	0.7198	0.0386	0.1962	0.1474	0.1374
13	6	5	3	0	0.9528	0.0104	0.1015	0.0722	0.0642	0.6915	0.0408	0.2016	0.1531	0.1445
14	7	5	3	0	0.9499	0.0111	0.1052	0.0747	0.0659	0.6866	0.0412	0.2027	0.1502	0.1404
15	8	5	3	0	0.9525	0.0106	0.1027	0.0728	0.0649	0.7073	0.0400	0.1999	0.1503	0.1411

Varying the latent space: quantum model in PennyLane + PyTorch @ 1000 epochs



	Experiments				Training					Validation				
	Lay	Lat	Tr	Xtr	TR2	TMSE	TRMSE	TMAE	TMAPE	VR2	VMSE	VRMSE	VMAE	VMAPE
Run														
0	3	8	0	1	0.9732	0.0062	0.0770	0.0581	0.0541	0.7139	0.0417	0.2034	0.1545	0.1478
1	3	7	1	1	0.9736	0.0061	0.0764	0.0579	0.0545	0.7350	0.0373	0.1928	0.1467	0.1460
2	3	6	2	1	0.9667	0.0076	0.0864	0.0643	0.0602	0.6953	0.0438	0.2083	0.1518	0.1463
3	3	5	3	1	0.9540	0.0103	0.1003	0.0731	0.0653	0.6770	0.0455	0.2126	0.1620	0.1499
4	3	4	4	1	0.9244	0.0160	0.1221	0.0879	0.0765	0.6189	0.0499	0.2211	0.1688	0.1593
5	3	3	5	1	0.9056	0.0194	0.1346	0.0980	0.0866	0.6106	0.0553	0.2332	0.1765	0.1642
6	3	2	6	1	0.8435	0.0309	0.1703	0.1205	0.1035	0.4838	0.0653	0.2533	0.1814	0.1662
7	3	1	7	1	0.7197	0.0522	0.2284	0.1521	0.1263	0.2278	0.0895	0.2991	0.2136	0.1878

Comparing results
(PennyLane vs PyTorch)

A more in-depth analysis of model training shows:

In training and validation, the CAE models are less sensitive to the size of their latent space.

The QAE models show a very definite differentiation of their performance in relation to the size of latent space.

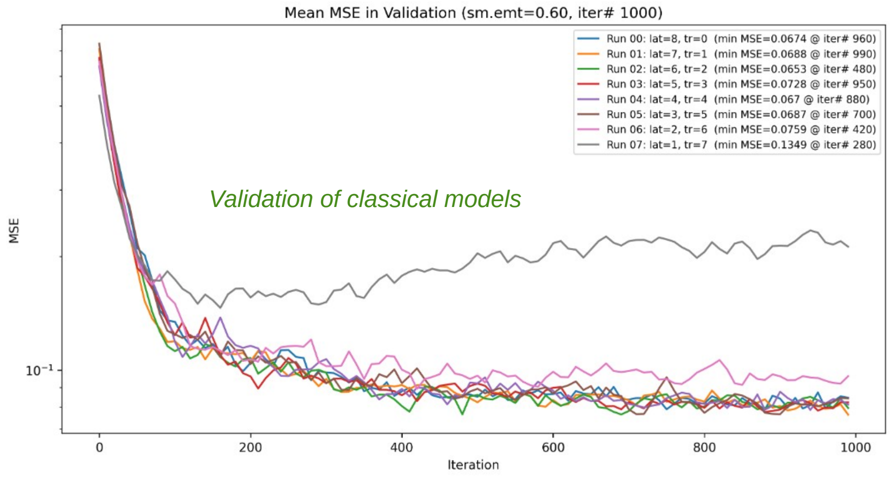
While both CAE and QAE still have the capacity for further learning (training), the CAE models reached their generalisation capacity (validation), while QAE models can still improve (validation).

CAE training curves show a lot of volatility, while QAE curves are smooth, indicating their training is more predictable.

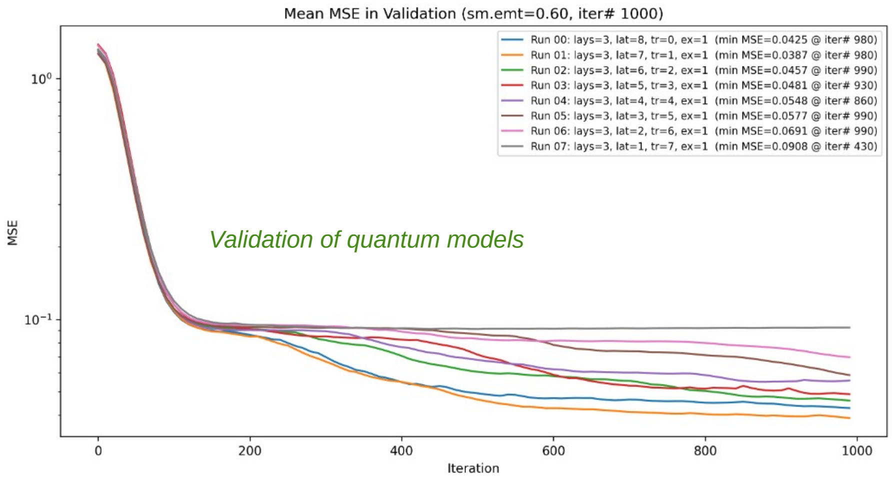
Note that the structural parameters for CAEs and QAEs may not be optimal, so the performance scores are indicative only.

USA beer sales (IRI)

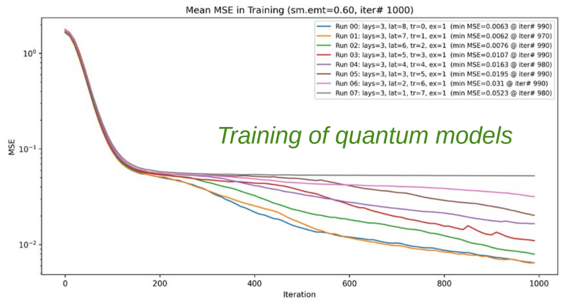
Mean MSE in validation (log y axis):
PyTorch @ 1000 epochs



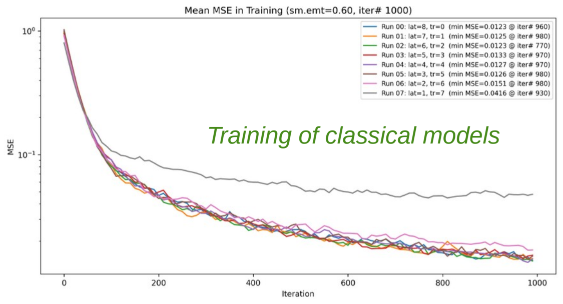
Mean MSE in validation (log y axis):
PennyLane + PyTorch @ 1000 epochs



Mean MSE in training (log y axis):
PennyLane + PyTorch @ 1000 epochs

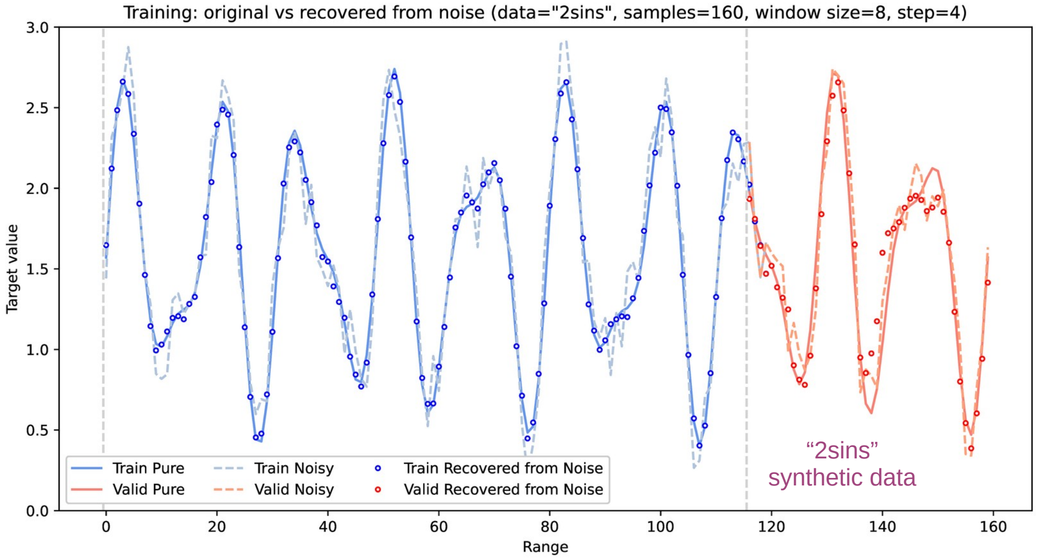


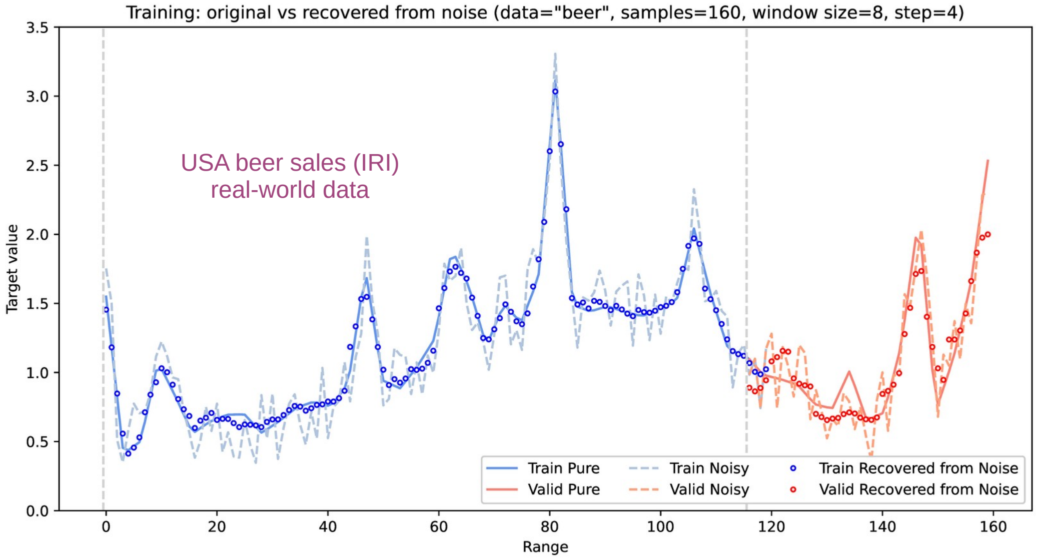
Mean MSE in training (log y axis):
PyTorch @ 1000 epochs

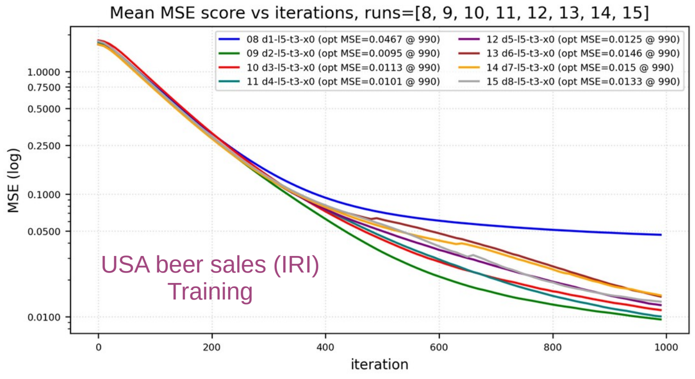


More results

PennyLane+ PyTorch



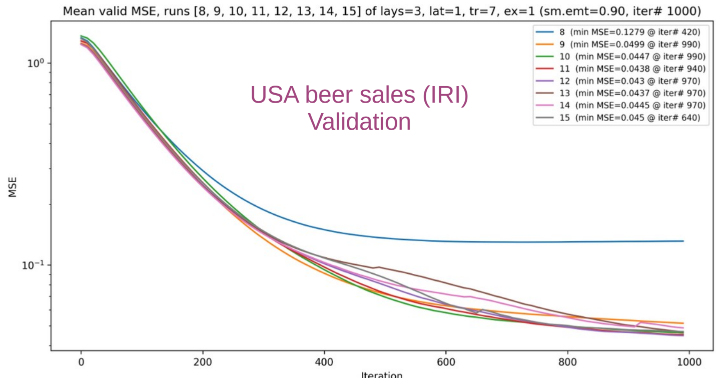


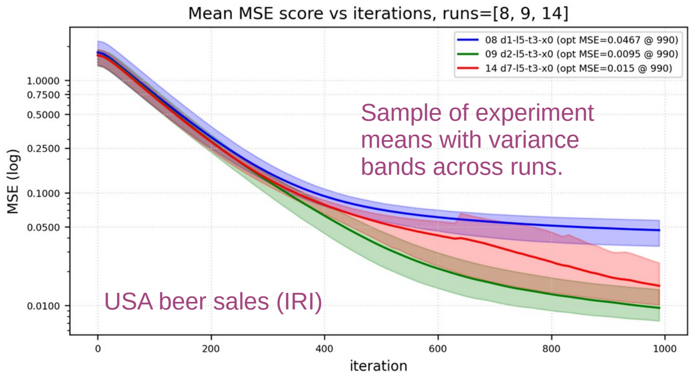


PennyLane performed consistently well on both synthetic (simple functions) and real-world data (beer sales).

After PennyLane training the QAE models (left) seem to fit the original data much closer (better denoising) than the Qiskit models.

Each model type was trained 10 times, randomly initialised. Mean performance with variance was considered.

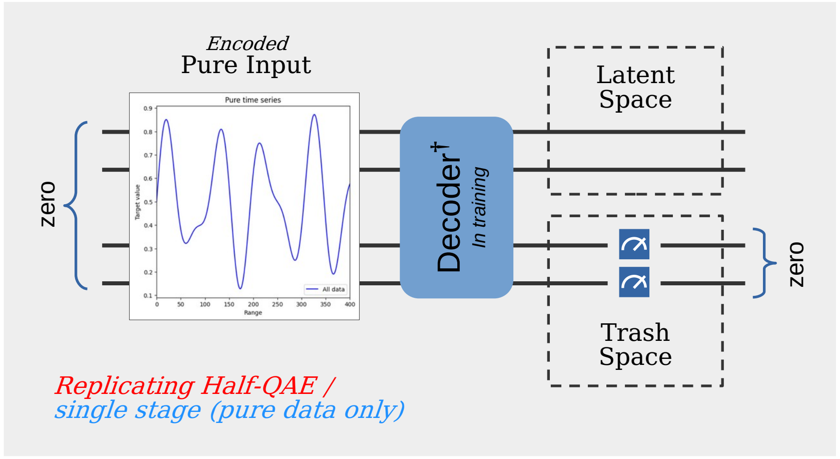


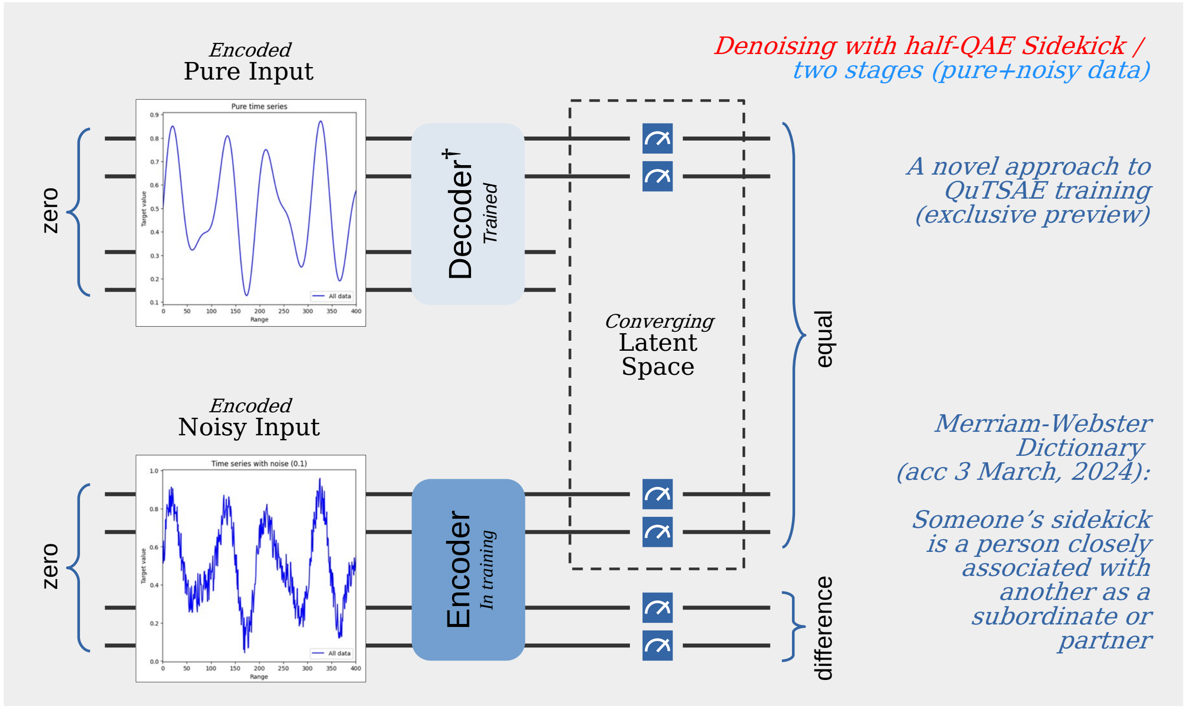


It is important to note that in training, Qiskit models converged within 1000 epochs.

PennyLane models (as can be observed here) still have the capacity to learn beyond the 1000 epochs.

Alternative Architectures

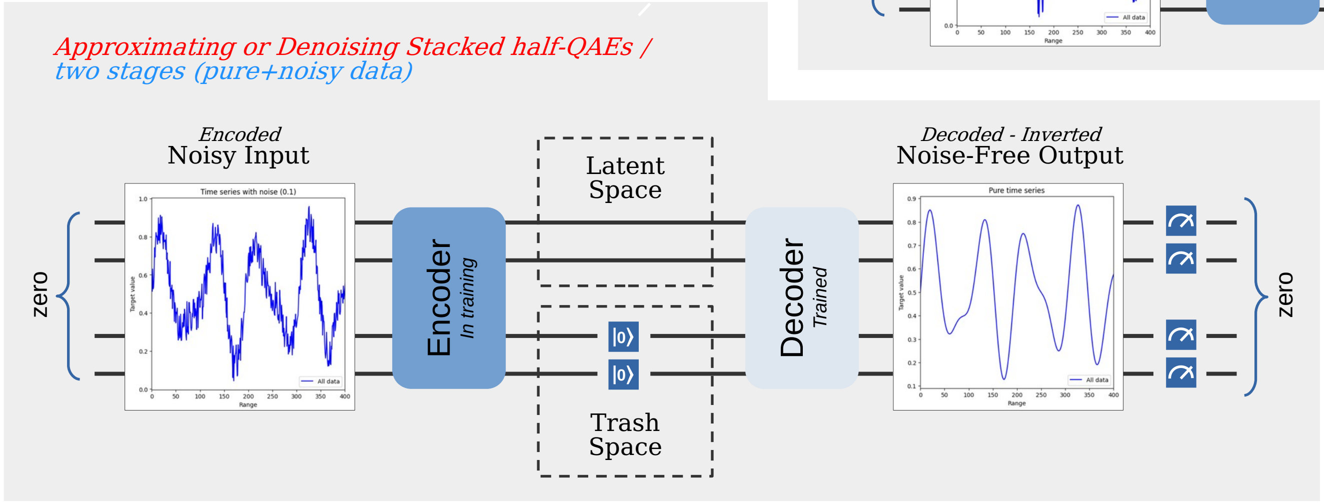




A novel approach to QuTSAE training (exclusive preview)

Merriam-Webster Dictionary (acc 3 March, 2024):

Someone's sidekick is a person closely associated with another as a subordinate or partner



We can train a pure QAE by training its half by converging trash info to zero, the other half is its inverse.

We can train a noisy half-QAE by stacking it with a pure half-QAE

We can also side-train a noisy half-QAE by converging its latent space to a pretrained pure half-QAE

Summary

Simulated VQA QAE models

Research insights

- We have discussed design decisions taken in the development of denoising quantum time series autoencoders
- Input encoding strategy needs to consider the output decoding strategy, i.e. measurements and their interpretation
- Methods of measuring and interpreting model's quantum state are essential for training and testing the model
- Ansatz architectural properties must fit the model function
- Ansatz circuit width, depth, the number of trainable parameters, as well as the required degree of freedom (extra qubits) determine the model performance
- Selection of a suitable cost function and an optimiser require experimentation
- Assessment of a quantum model performance requires statistical analysis!
- QAE pure quantum models are merely approaching the performance of classical models
- On simulators, hybrid quantum-classical models show better performance than either pure quantum or pure classic models
- However, hybrid models lose their quantum advantage when executed on quantum machines
- Development of quantum models can take advantage of such model features that are missing from the classical systems
- The majority of insights derived from this study are applicable to the design of many other QML models

Current work

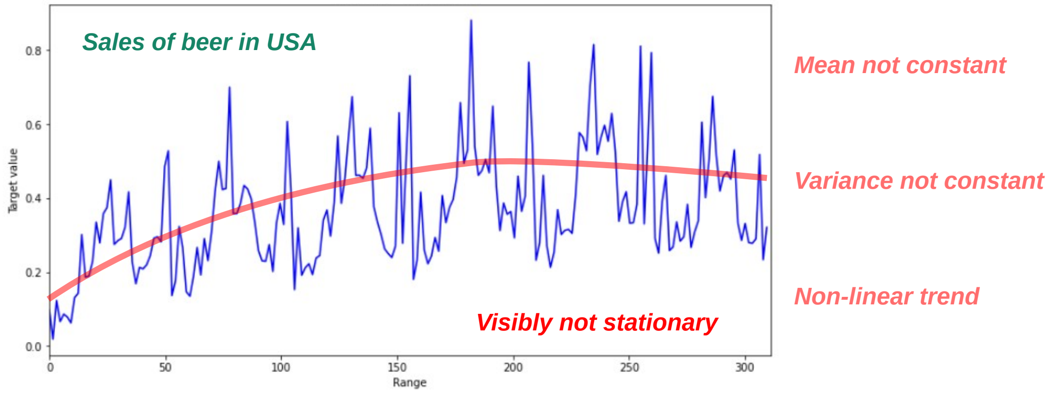
- Introduction of tighter integration between classical and quantum ML methods, which seem to result in more effective model optimisation
- Investigation of pure-quantum QAE, which can be trained effectively to improve validation scores.

Thank you!

Any questions?

Appendix:
Concepts in TS Analysis

- Time series analysis aims to *identify patterns* in historical time data and to *create forecasts* of what data is likely to be collected in the future
- Applications include heart monitoring, weather forecasts, machine condition monitoring, etc.
- Times series analysis is well established with *excellent tools* and *efficient methods*, yet some organisations aim to improve them further
- Time series must have an *unique index* - a time-stamp sequencing the series
- 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)
- 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
- *Quantum time series analysis (QTSA) is a promising approach to time series analysis and forecasting!*



Appendix:
Quantum Neural Networks

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.

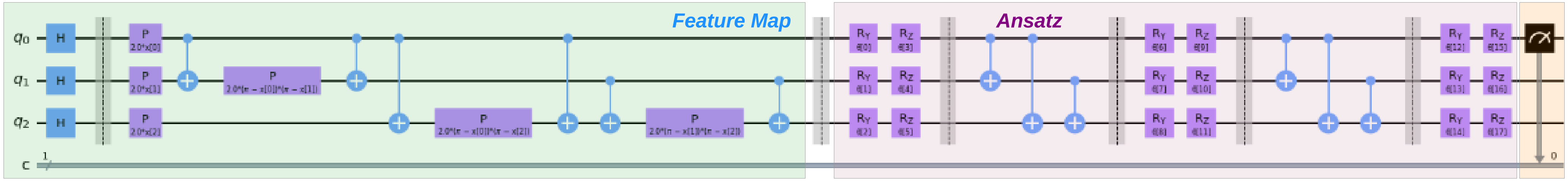
Schreiber, Amelie. "Quantum Neural Networks for FinTech." Medium, May 8, 2020. https://towardsdatascience.com/quantum-neural-networks-for-fintech-dddc6ac68dbf.

- 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

Pattern Matching

- 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

VQR Model


Feature Map
Ansatz
q0
q1
q2
c
1
H
P 2.0*x[0]
P 2.0*x[1]
P 2.0*x[2]
P 2.0*(π − x[0])*(π − x[1])
P 2.0*(π − x[0])*(π − x[2])
P 2.0*(π − x[1])*(π − x[2])
RY θ[0]
RY θ[1]
RY θ[2]
RZ θ[3]
RZ θ[4]
RZ θ[5]
RY θ[6]
RY θ[7]
RY θ[8]
RZ θ[9]
RZ θ[10]
RZ θ[11]
RY θ[12]
RY θ[13]
RY θ[14]
RZ θ[15]
RZ θ[16]
RZ θ[17]
0