Secrets revealed in this session:

To understand the process of developing complex quantum models

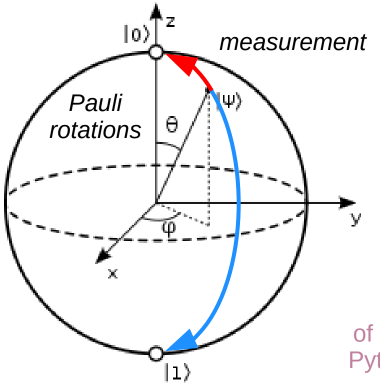
QNN overview
Building simple QNNs (Q-MLP)
Data encoding strategies
Measurements and their interpretation
Training QNN models
Measuring model performance
Barriers to model learning
Overcoming training difficulties
QNN models vs classical Nns
    QCNN, QAE, QGAN, QLSTM, QGNN, QTrans, ...
QNN applications
    PennyLane demo
Summary

Quantum Neural Networks

Inspired by the brain, executed with lightning speed

Jacob L. Cybulski

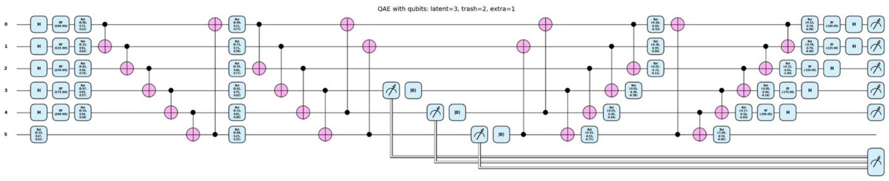
Enquanted, Melbourne, Australia



We will assume some knowledge of QC and QML plus Python programming

Presenter

Jacob Cybulski
quantum@jacobcybulski.com

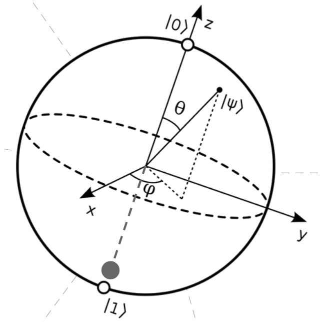


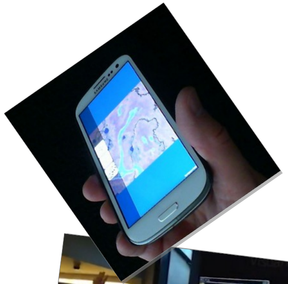
Founder
Researcher
Consultant
Author
at Enquanted

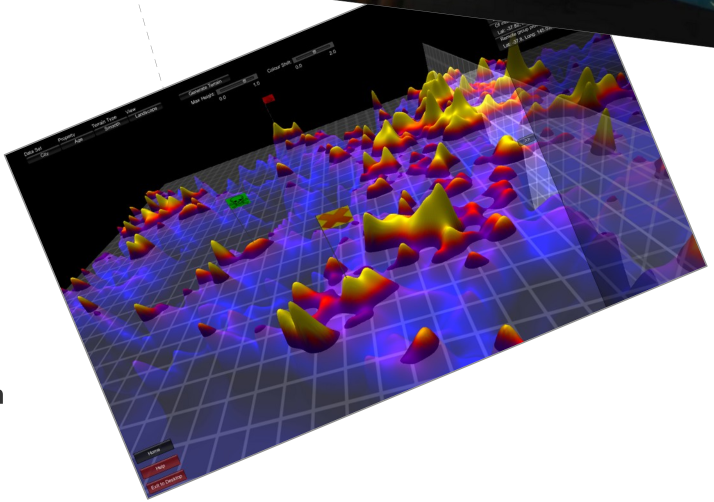
Melbourne
Australia

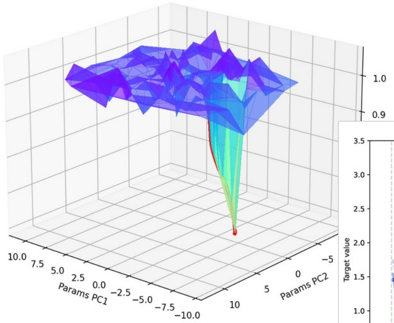
Research
- Quantum computing
- Quantum machine learning
- Quantum time series analysis and anomaly detection
- Classical machine learning
- Data visualisation

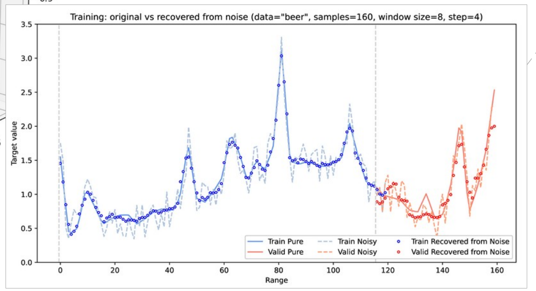
Personal
- Recreational cycling
- Reading science and Sci-Fi
- Quantum challenges and hackathons











Research collaboration and supervision of research students in QC + QML

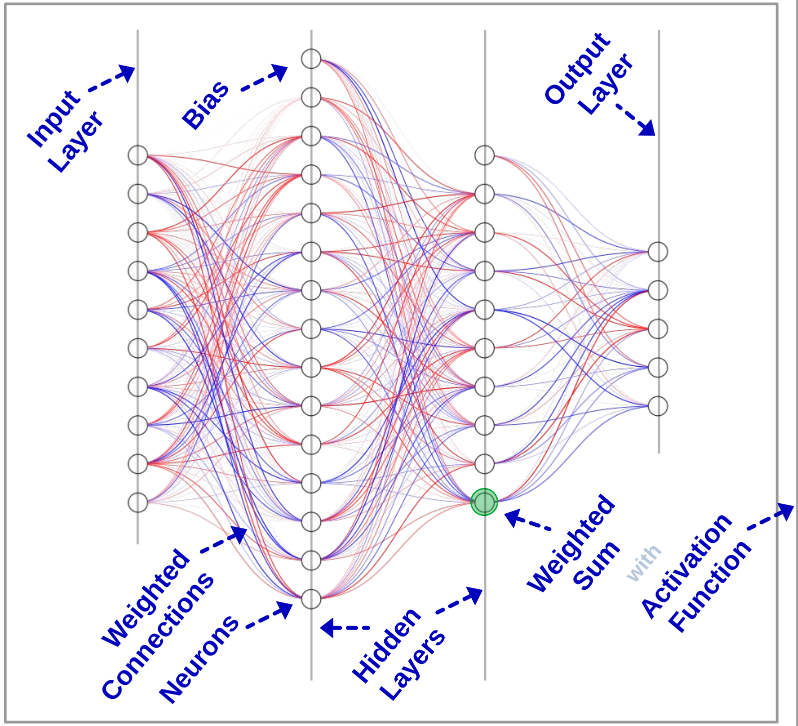
Neural Networks

A class of complex ML models

MLP = a simple neural network

capable of learning any “smooth” function
learns to associate inputs with outputs

- *Multi-Layer Perceptrons* (MLP) take numerical input and produce numerical output
- MLPs are structured into layers
- *Layers* consist of neurons
- *Neurons* hold activation - value in range [-1,+1]
- *Weighted links* connect neurons of adj. layers
- *Activation* is a weighted sum of activations of neurons from the previous layer
- *Bias* is a value added to the sum
- *Activation function* is applied to the sum to scale the result back to the interval [-1, 1]
- *Optimisation* is the process to identify optimum weights and biases, it is commonly iterative
- *Optimisation aims* to reduce cost or aggregated loss, a distance between the calculated and expected results



- This process can be accelerated by using specialised hardware, e.g. GPUs or TPUs
- Other NNs: CNN, AE, GAN, LSTM + QNNs

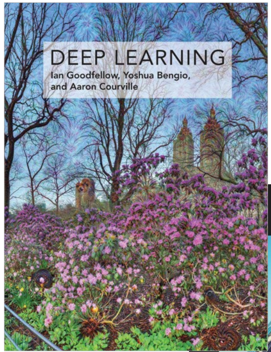
Name	Plot	Function, $g(x)$
Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Soboleva modified hyperbolic tangent (smht)		$\operatorname{smht}(x) \doteq \frac{e^{ax} - e^{-bx}}{e^{cx} + e^{-dx}}$
Rectified linear unit (ReLU)[10]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[2]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$
Softplus[11]		$\ln(1 + e^x)$
Exponential linear unit (ELU)[12]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Scaled exponential linear unit (SELU)[13]		$\lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$
Leaky rectified linear unit (Leaky ReLU)[14]		$\begin{cases} 0.01x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$
Parametric rectified linear unit (PReLU)[15]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α
Sigmoid linear unit (SiLU,[2] Sigmoid shrinkage,[16] SiL,[17] or Swish-1[18])		$\frac{x}{1+e^{-x}}$
Gaussian		e^{-x^2}

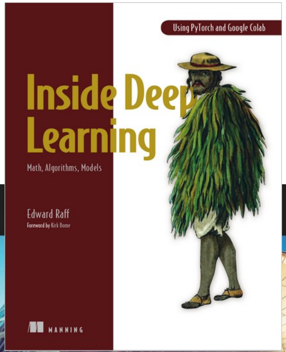
Wikipedia

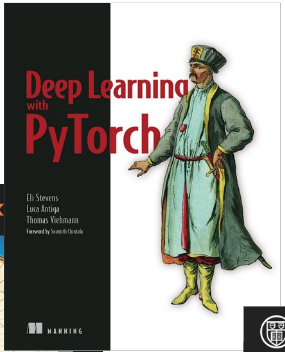
Recommended reading

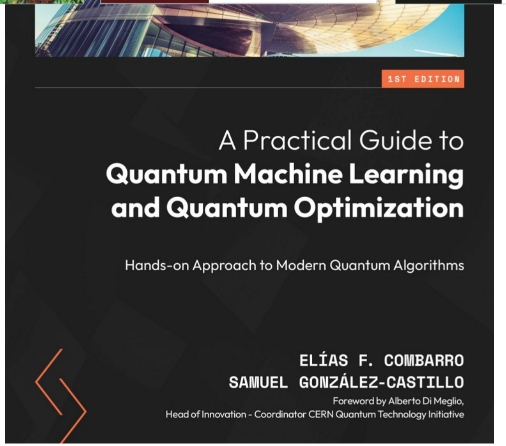
on QNN + Deep Learning

Classical Neural Networks / Deep Learning



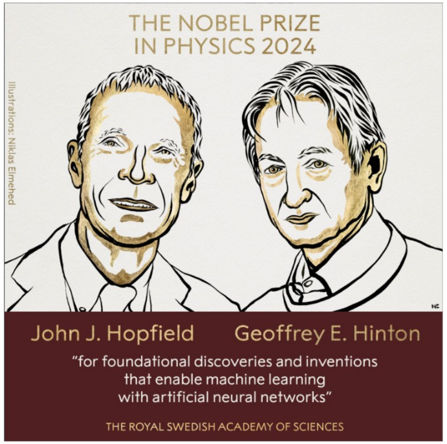


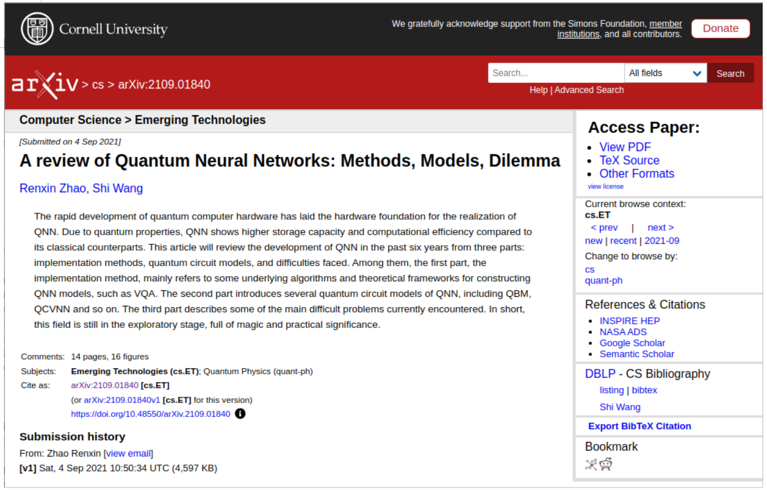




Chapter 10: QNN

Neural Networks Fathers





Cornell University

We gratefully acknowledge support from the Simons Foundation, member institutions, and all contributors. Donate

arXiv > cs > arXiv:2109.01840

Computer Science > Emerging Technologies

[Submitted on 4 Sep 2021]

A review of Quantum Neural Networks: Methods, Models, Dilemma

Renxin Zhao, Shi Wang

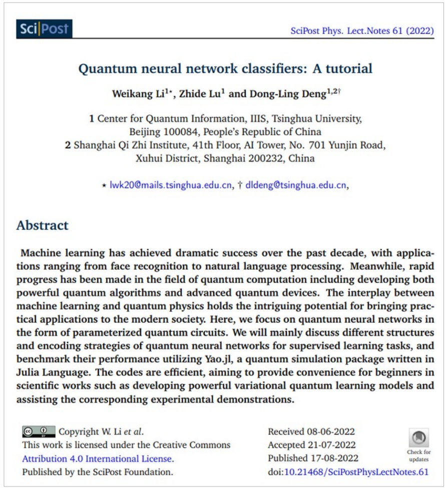
The rapid development of quantum computer hardware has laid the hardware foundation for the realization of QNN. Due to quantum properties, QNN shows higher storage capacity and computational efficiency compared to its classical counterparts. This article will review the development of QNN in the past six years from three parts: implementation methods, quantum circuit models, and difficulties faced. Among them, the first part, the implementation method, mainly refers to some underlying algorithms and theoretical frameworks for constructing QNN models, such as VQA. The second part introduces several quantum circuit models of QNN, including QBM, QCVNN and so on. The third part describes some of the main difficult problems currently encountered. In short, this field is still in the exploratory stage, full of magic and practical significance.

Comments: 14 pages, 16 figures
Subjects: Emerging Technologies (cs.ET); Quantum Physics (quant-ph)
Cite as: arXiv:2109.01840 [cs.ET]
(or arXiv:2109.01840v1 [cs.ET] for this version)
https://doi.org/10.48550/arXiv.2109.01840

Submission history
From: Zhao Renxin [view email]
[v1] Sat, 4 Sep 2021 10:50:34 UTC (4,597 KB)

Access Paper: View PDF, TeX Source, Other Formats

Variety of QNN Types



SciPost Phys. Lect.Notes 61 (2022)

Quantum neural network classifiers: A tutorial

Weikang Li[1*], Zhide Lu[1] and Dong-Ling Deng[1,2†]

1 Center for Quantum Information, IIIS, Tsinghua University, Beijing 100084, People's Republic of China
2 Shanghai Qi Zhi Institute, 41th Floor, AI Tower, No. 701 Yunjin Road, Xuhui District, Shanghai 200232, China

* lwk20@mails.tsinghua.edu.cn, † dldeng@tsinghua.edu.cn,

Abstract

Machine learning has achieved dramatic success over the past decade, with applications ranging from face recognition to natural language processing. Meanwhile, rapid progress has been made in the field of quantum computation including developing both powerful quantum algorithms and advanced quantum devices. The interplay between machine learning and quantum physics holds the intriguing potential for bringing practical applications to the modern society. Here, we focus on quantum neural networks in the form of parameterized quantum circuits. We will mainly discuss different structures and encoding strategies of quantum neural networks for supervised learning tasks, and benchmark their performance utilizing Yao.jl, a quantum simulation package written in Julia Language. The codes are efficient, aiming to provide convenience for beginners in scientific works such as developing powerful variational quantum learning models and assisting the corresponding experimental demonstrations.

Copyright W. Li et al. This work is licensed under the Creative Commons Attribution 4.0 International License. Published by the SciPost Foundation.

Received 08-06-2022
Accepted 21-07-2022
Published 17-08-2022
doi:10.21468/SciPostPhysLectNotes.61

QNN in Julia

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.
Schreiber, Amelie. “Quantum Neural Networks for FinTech.” Medium, May 8, 2020.

Quantum Neural Networks

Specifically Quantum MLPs

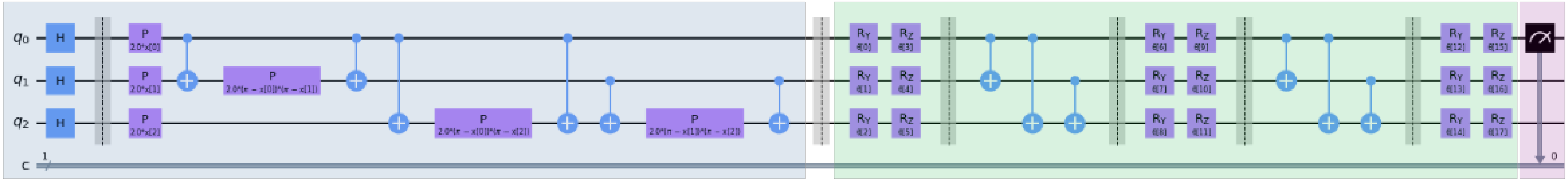
- The QNN variational model is typically represented by a quantum circuit of three components, i.e.
  - ➔ *feature map* encoding QNN’s classical input data and preparing the circuit’s quantum state
  - ➔ *ansatz* consisting of several layers of trainable parameters (Pauli rotations), responsible for quantum state processing and transformation
  - ➔ *measurement* of the circuit’s quantum state, which can subsequently be interpreted as QNN’ classical output
- QNNs can be trained with variational quantum algorithms and a wide range of classical optimisers.
- Pure quantum training strategies are also possible.

QNNs

- can deal with highly complex computation (as QM)
- can deal with large volume of data (as NN)
- can process entire probabilistic distributions of values (superposition) and utilise parameters space of exponential size (entanglement)
- require repeated execution to produce output
- are missing some efficiency of classical NNs (non-linear activations and regularisation strategies)
- are difficult to process on quantum simulators of limited computational capacity
- need experimentation and extensive data preparation – there is no magic in quantum tech!

As per a VQA Model

Sample Feature Map (Input) *Sample Ansatz (Processing)* *Sample Measurement (Output)*

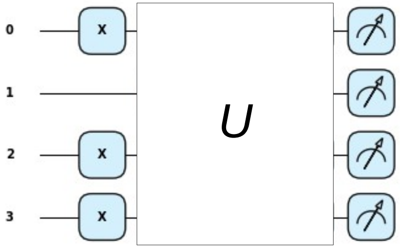


Data embedding:
Basis encoding and decoding

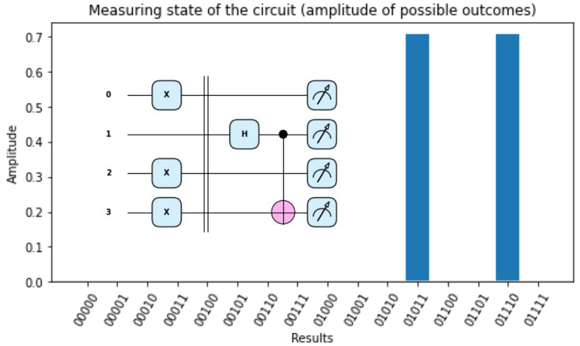
There are many different approaches to quantum data encoding and decoding that are suitable for QNN, e.g. basis, angle and amplitude embedding.

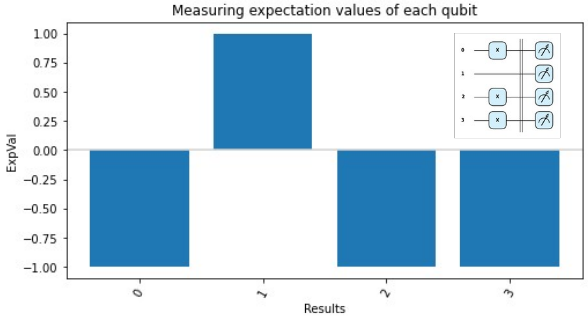
Basis embedding is the commonly used strategy for quantum encoding and decoding of integer numbers, where:

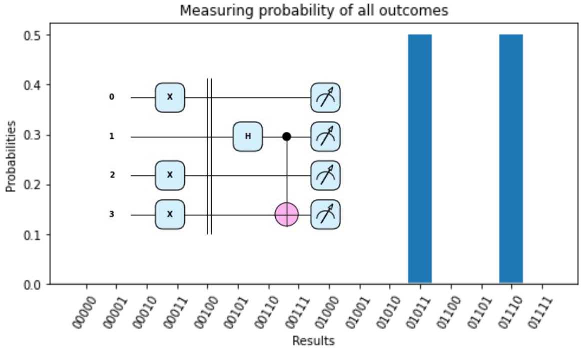
- qubits act as bits of the encoded numbers
- circuit state can be interpreted as bits of the numeric value on output
- application needs a single (or very few) integer value on input and output

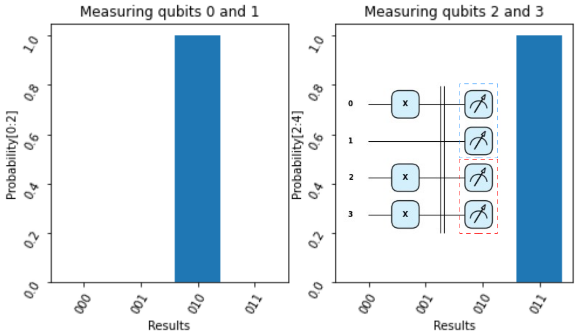


Encoding and measuring number 11









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.

Data embedding:
Angle encoding and decoding

Angle embedding represents numeric values as properties of qubit state rotation (angle, amplitude or probability)

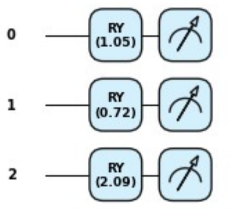
The rotation operators are the basic quantum operation

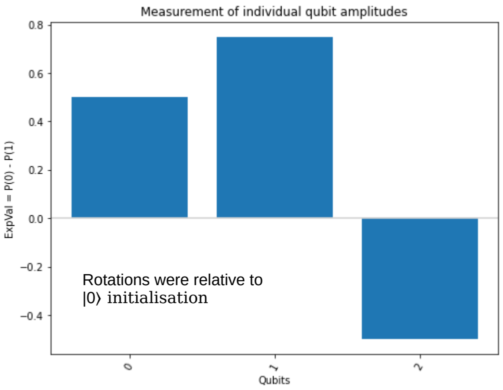
Encoding rotations are performed around x, y, z axes of the Bloch sphere (multiple values per qubit are possible)

Rotations are relative to a specific qubit state, e.g. |0⟩

Input encoding can be repeated across the circuit, called reuploading, which improves the model performance

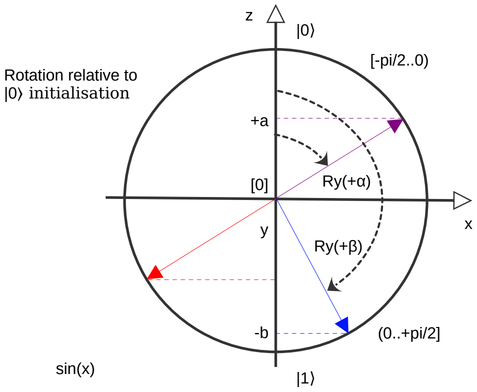
As training will place qubit states in areas x < 0 and around the z axis, measurements may not distinguish these states from “pure” x > 0 and z = 0.



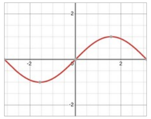


Input
Values entered: [np.arccos(0.5), np.arccos(0.75), np.pi-np.arccos(0.5)]
Ry angles used: [1.047, 0.723, 2.094]

Measurements
Probabilities: [[0.25, 0.75], [0.562, 0.438], [0.25, 0.75]]
Amplitudes: [0.5, 0.75, -0.5]



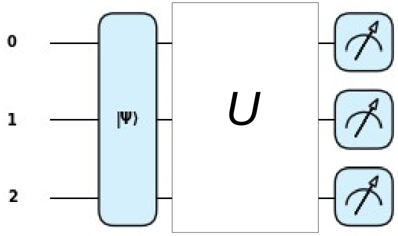
sin(x)

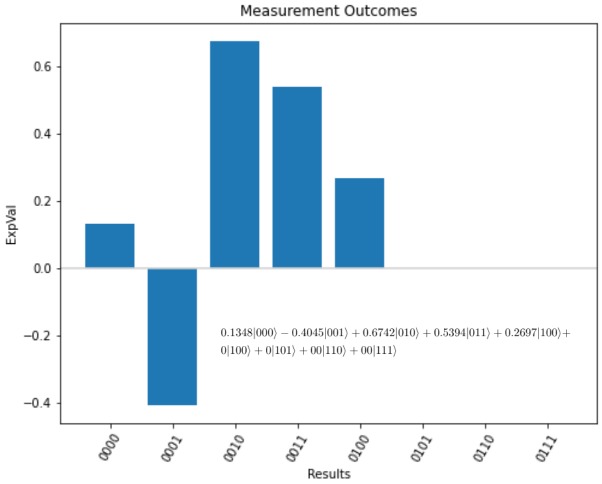


Sutor, R.S., 2024. Dancing with Qubits, Second Edition. Packt Pub.

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.

Data embedding:
Amplitude encoding and decoding





There are many other methods of data encoding, e.g. QRAM, time-evolution, or dense-angle, or Hamiltonian encoding.

Many of these methods are offered as ready-made feature maps.

Qiskit feature maps include:

- ZfeatureMap
- ZZFeatureMap
- PauliFeatureMap

Amplitude embedding is one of the most useful encoding / decoding strategies

Unless supported by the quantum platform, it is considered the most difficult (see Sutor 2024)

In amplitude encoding, each data point is encoded as expectation value of multi-qubit measurement of all qubits' states

This way, we can embed 2^{qubits} numbers into a circuit!

Consider a vector:

$v = [0.1, -0.3, 0.5, 0.4, 0.2]$,

which needs to be normalised by the vector length:

$\mathrm{sqrt}(0.1^2+(-0.3)^2+0.5^2+0.4^2+0.2^2)$,

which results in a new vector (approximately):

$v' \approx [0.13484\ -0.40452\ \ 0.6742\ \ \ 0.53936\ \ 0.26968]$.

To encode 5 amplitudes in a quantum circuit, we need at least 3 qubits. Thus, resulting in the following encoding:

$$0.1348|000\rangle - 0.4045|001\rangle + 0.6742|010\rangle + 0.5394|011\rangle + 0.2697|100\rangle + 0|100\rangle + 0|101\rangle + 00|110\rangle + 00|111\rangle$$

We will rely on PennyLane and Qiskit to generate quantum gates for this circuit ...

Commonly used
measurements and interpretation

Model measurement and interpretation of results share their fundamental concepts and methods with data encoding.

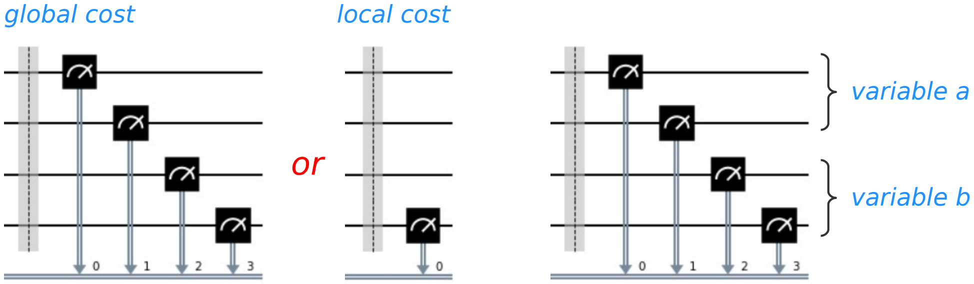
There are many ways of obtaining the outcome of a circuit execution, e.g. we can measure:

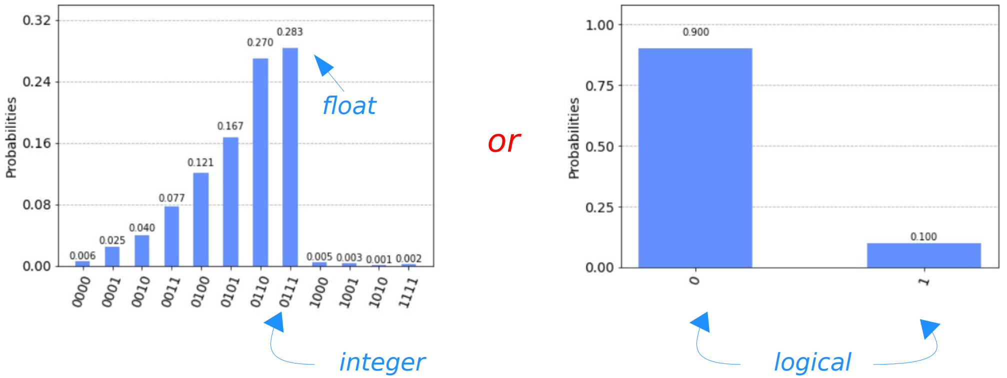
- all qubits (global cost)
- a few qubits (local cost)
- groups of qubits
- as counts of repeated measurements
- as probabilities of |0⟩ and |1⟩
- as expectation values, P(0)-P(1)
- as variance, etc.

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

- as a binary outcome:
  single qubit measurement
- as an integer:
  multi-qubit measurement
- as a continuous variable:
  expectation value of a specific outcome

Circuit state measurement has an impact on the calculation of the loss/cost function

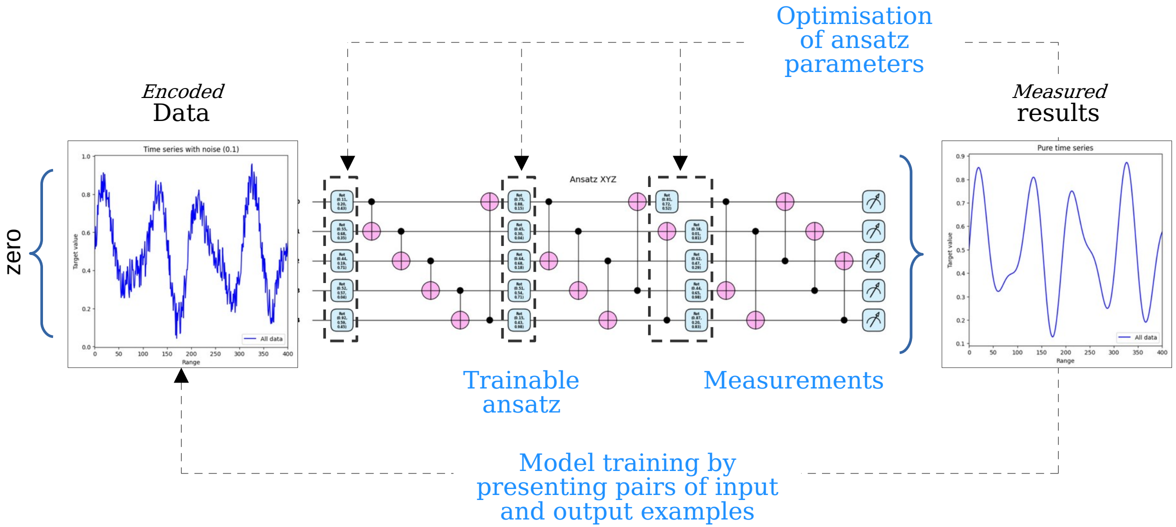




Probability distribution of measurements can be further interpreted, e.g. we could check parity of the probability kets (e.g. |110⟩ is even, while |111⟩ is odd), add all even and odd probabilities respectively, and treat the result as a logical measurement.

Training of Quantum Neural Networks

- QNN training needs a loss / cost function and an optimiser of the model parameters
- The loss function and optimiser can either be pure quantum or hybrid
- Pure quantum approach often relies on quantum adiabatic or quantum annealing optimisation, and Grover-like amplitude amplification
- A hybrid approach uses variational quantum algorithms (VQA), and relies on the QNN execution on a quantum machine, and its parameters optimisation conducted on a classical machine
- Hybrid training of QNNs is identical to training classical NN models



Optimisation example

Gradient descent

Consider the house price (y) as a function of the size of its front yard (x). Let us consider all models ($f_{a,b}$) to estimate the house prices by the formula:

$$y = f_{a,b}(x) = a * x + b$$

Each model is parameterised by a and b, and can fit a sample A={x, y} of house training data (Ames real estate).

For each house (x, y), a model $f_{a,b}$ will make some error (loss). For all houses in A it will accumulate these errors as a single value (cost), e.g. MAE (mean absolute error).

The cost of each model is a point in a 3D space

a x b x MAE

All such points form a “cost” surface.

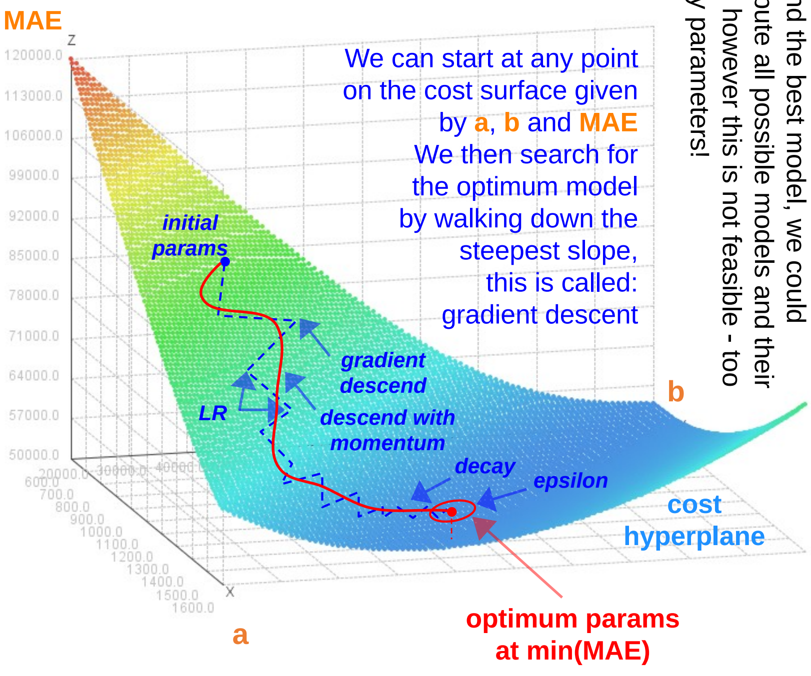
The shape of such a surface we call the cost landscape.

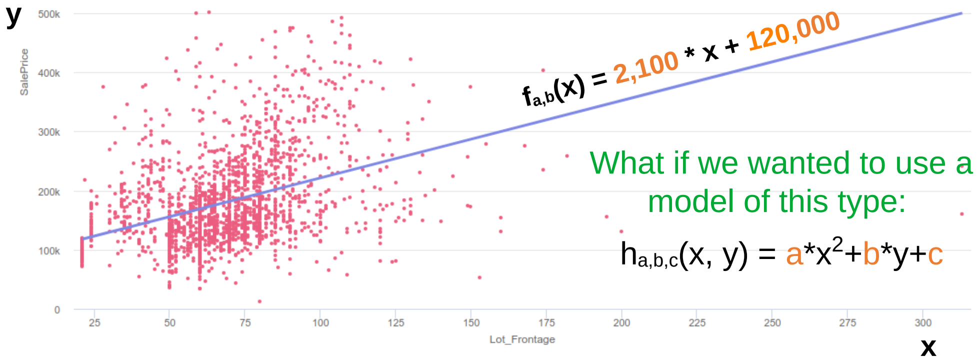
When a model has many parameters, the cost surface is called a hyperplane.

Now we will look for the best model, i.e. the model which generates the lowest MAE

To find the best model, we could compute all possible models and their cost, however this is not feasible - too many parameters!

We can start at any point on the cost surface given by a, b and MAE. We then search for the optimum model by walking down the steepest slope, this is called: gradient descent





What if we wanted to use a model of this type:

$$h_{a,b,c}(x, y) = a*x^2+b*y+c$$

The optimiser controls this process via its hyper-parameters, i.e. parameters of the gradient descent itself:

learning rate
momentum
decay
epsilon

By using gradient descent, the optimum cost (and thus the model), was found at:

A=1060 (Lot_Frontage)
B=90000 (Intercept)
MAE=53473.097 (Error)

Measuring QNN "quality"

An area of Jacob's research

Abbas, A., Sutter, D., Zoufal, C., Lucchi, A., Figalli, A., Woerner, S., 2021. The power of quantum neural networks. Nature Comput Sci 1, 403–409.

Abbas, A., Sutter, D., Figalli, A., Woerner, S., 2021. Effective dimension of machine learning models, arXiv:2112.04807.

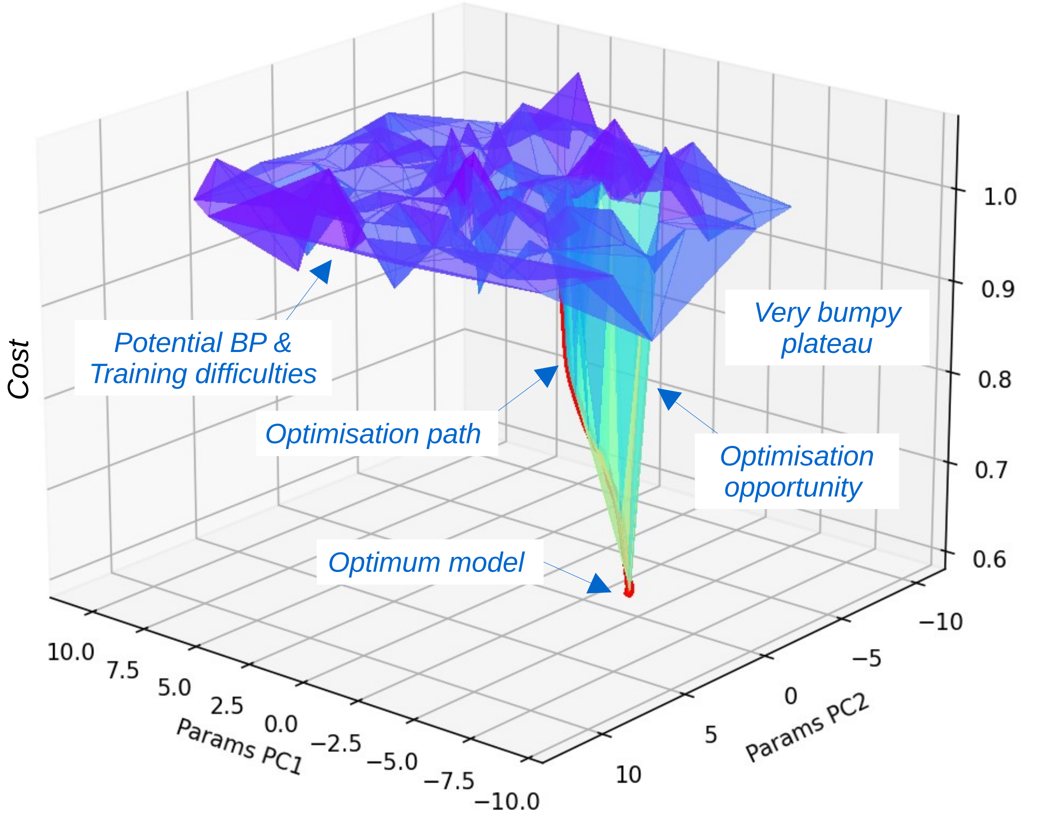
Cybulski, J.L., Nguyen, T., 2023. Impact of barren plateaus countermeasures on the quantum neural network capacity to learn, Quantum Information Processing 22, 442.

- The common way of measuring QNN quality is to measure its ability to *generalise* beyond training data.
- This can be assessed by using separate data sets, i.e. *validation and test data*, as well as some metric, e.g. MSE, MAE, accuracy, cross-entropy, etc.
- At different points in training, we could also measure the *model capacity*:
  - ➔ *storage capacity* measured in bits and bytes of information (Little and Shaw 1978, Newman 1988)
  - ➔ *network potential* for storing input-output patterns (Gardner and Derrida, 1988, Gardner, 1988)
  - ➔ *requisite complexity* as the ability to accurately approximate a given function (Hornik, 1991)
  - ➔ *optimal brain damage* as the ability to accommodate removal of parameters without adversely affecting information contents (LeCun et al., 1989)
  - ➔ *capacity to learn* as the ability to generalise from the previously learnt training data
- *Capacity to learn* has been explored and formalised in a number of different ways:
  - ➔ *VC-dimension* as the set of functions the neural network could represent depending on the size of the training set and its tolerance for the error rates (Vapnik and Chervonenkis, 1971)
  - ➔ *effective VC-dimension* which takes into account not only the size of the training data and error rates, but also the probability distribution of its measurements (Vapnik, Levin & Le Cun, 1994)
  - ➔ *volume of the cost gradient geometry* emerging from the network's optimisation, and which can be defined using Fisher Information Matrix (Karakida, 2020)
  - ➔ *the number of independent pure quantum states* that can be represented (Lewenstein et al, 2021)
  - ➔ *effective quantum dimension* as the relationship between the model geometry, expressive power and redundancy, effectiveness of its training and initialisation strategy (Abbas et al, 2021)

Barren plateaus

An area of Jacob's research

Barren plateaus (BPs) are large "flat" areas in the quantum model's cost landscape, which impede model optimisation.



- QNNs have similar training difficulties as Nns
- BPs are related to vanishing gradients in NNs
- BPs presence does not mean the model is bad, its training is just more difficult
- BPs are the natural feature of measurements in high dimensional space of model parameters
- BPs do not just "exist", they emerge in training
- BPs are commonly flat, however, their surface may become rough and bumpy due to noise
- BP countermeasures can make your model worse
- There exist well-known causes of BPs and there are well-known BP countermeasures, e.g.
  1) use fewer qubits / layers / parameters
  2) use local cost functions
  3) beware of random params initialisation
  4) use BP-resistant model design (e.g. layerwise)
  5) use BP-resistant models (e.g. QCNNs)

Cybulski, J.L., Nguyen, T., 2023. "Impact of barren plateaus countermeasures on the quantum neural network capacity to learn", Quantum Inf Processing 22, 442.

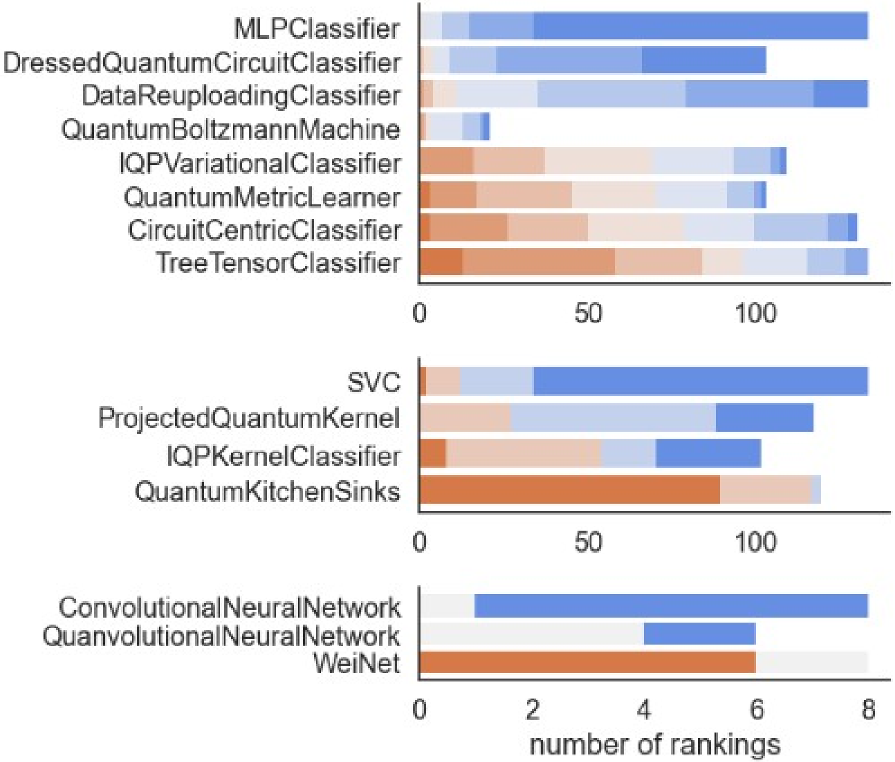
Cerezo, M., Sone, A., Volkoff, T., Cincio, L., Coles, P.J., 2021. Cost function dependent barren plateaus in shallow parametrized quantum circuits. Nat Commun 12, 1791.

Grant, E., Wossnig, L., Ostaszewski, M., Benedetti, M., 2019. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. Quantum 3, 214.

Skolik, A., McClean, J.R., Mohseni, M., van der Smagt, P., Leib, M., 2021. Layerwise learning for quantum neural networks. Quantum Mach. Intell. 3, 5.

Quantum vs Classical:
Will QML give an advantage?

- Recent benchmarking show that classical models outperform quantum models (Bowles, et al, 2024)
- Quantum advantage over classical models cannot be easily verified, and experiments cannot be reproduced!
- Dressed models (NNs with a quantum layer) perform well, yet it cannot be proven it is due to the quantum element
- Data re-uploading genuinely improves the quantum model's performance
- The nature of training data influences quantum models performance far more than the classical models
- Lessons learnt:
  - when introducing a quantum method to machine learning, we need to carefully establish in what way this may alter or benefit the better established classical approaches
  - rather than adapting a classical model, we may need to introduce a unique quantum approach to model creation and optimisation!
- QNNs and QML are still in their early development - the new field is very exciting and very frustrating!



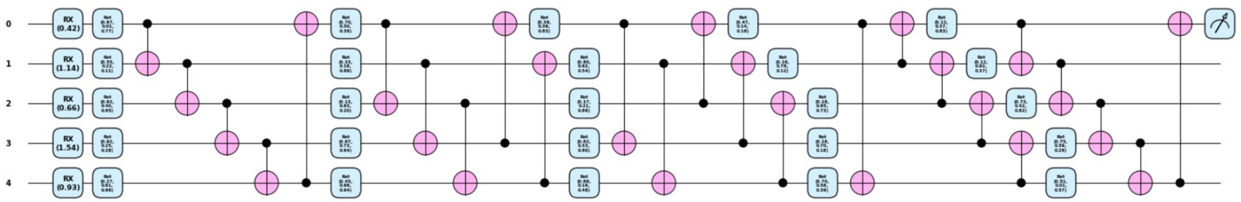
(rankings: blue/best to red/worst)

Bowles J, Ahmed S, Schuld M. Better than classical? The subtle art of benchmarking quantum machine learning models. arXiv; 2024 [accessed 2024 Oct 8]

For PDF of slides and links to GitHub code, see:
https://jacobcybulski.com/ (Presentations menu)

Demo:

Estimate diabetes progression one year after baseline



Which estimator is better?
Which could still improve?

Would this change if we were running the model training on a quantum machine?

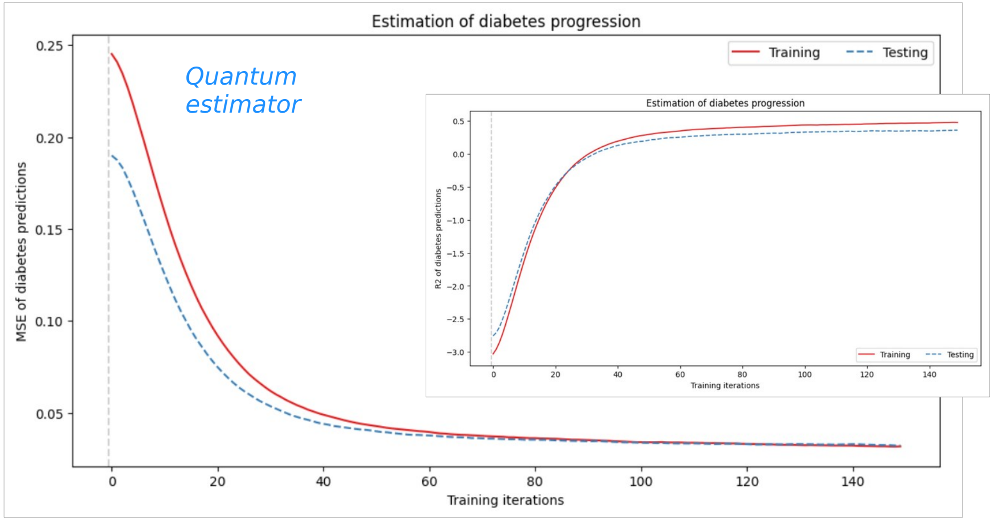
```
devices = cpu + lightning.qubit
samples = 296, features = 5, params = 75, epochs = 150
training: cost = 0.0306 @ 0141, r2 = 0.4977 @ 0141
testing:  cost = 0.0309 @ 0148, r2 = 0.3891 @ 0148
elapsed time = 3526sec (00:58:46)
```

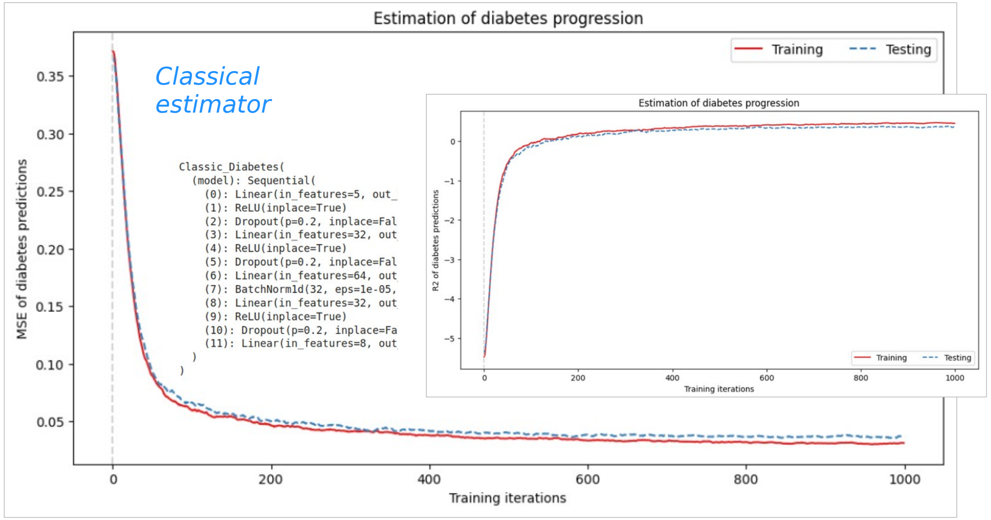
```
device = cpu
samples = 296, features = 5, params = 4721, epochs = 1000
training: cost = 0.0278 @ 0852, r2 = 0.5147 @ 0852
testing:  cost = 0.0304 @ 0980, r2 = 0.4708 @ 0980
elapsed time = 3sec (00:00:03)
```

```
Model training started

  0 (000024 sec): Loss 0.2452   R2 -3.0295
  7 (000189 sec): Loss 0.0971   R2 -0.5967
 14 (000354 sec): Loss 0.0596   R2 0.0204
 21 (000519 sec): Loss 0.0499   R2 0.1802
 28 (000684 sec): Loss 0.0455   R2 0.2517
 35 (000848 sec): Loss 0.0421   R2 0.3077
 42 (001013 sec): Loss 0.0404   R2 0.3354
 49 (001178 sec): Loss 0.0388   R2 0.3618
 56 (001343 sec): Loss 0.0385   R2 0.3669
 63 (001507 sec): Loss 0.0371   R2 0.3904
 70 (001671 sec): Loss 0.0359   R2 0.4102
 77 (001835 sec): Loss 0.0347   R2 0.4293
 84 (002000 sec): Loss 0.0349   R2 0.4261
 91 (002164 sec): Loss 0.0343   R2 0.4368
 98 (002329 sec): Loss 0.0329   R2 0.4586
105 (002493 sec): Loss 0.0324   R2 0.4673
112 (002657 sec): Loss 0.0333   R2 0.4525
119 (002822 sec): Loss 0.0313   R2 0.4859
126 (002986 sec): Loss 0.0312   R2 0.4870
133 (003151 sec): Loss 0.0316   R2 0.4811
140 (003315 sec): Loss 0.0321   R2 0.4727
147 (003479 sec): Loss 0.0308   R2 0.4935

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





Thank you!

Any questions?

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