doitnow
HPC Services

# QML with Qiskit

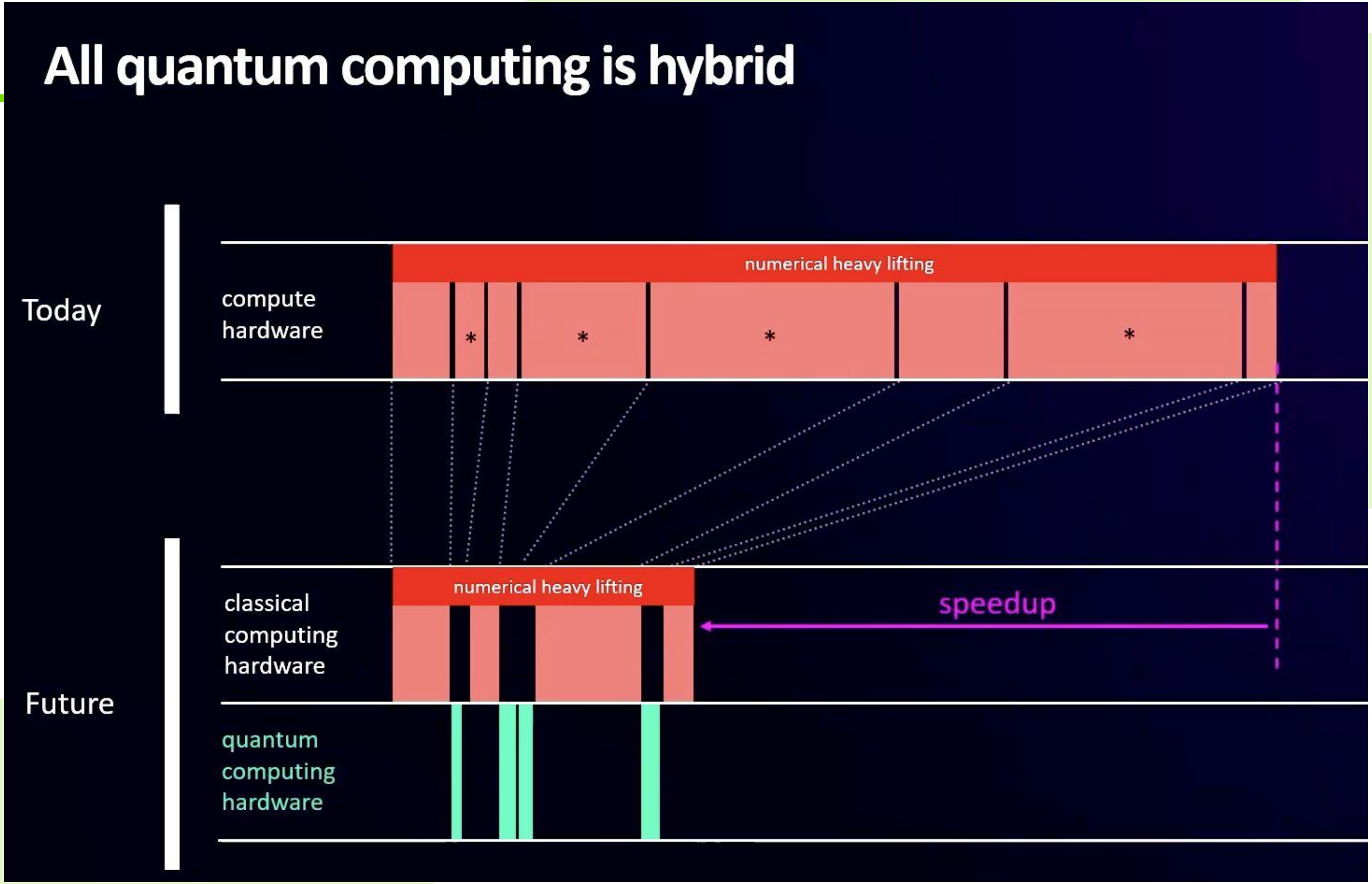
Lourens van Niekerk

doitnow HPC Services

# Is quantum machine learning better?

- We do not know yet where and how for sure
- Many cases of better results are with advantageous datasets
- When is it better?
  - Faster training time for good, small datasets - Finance
  - Better classification in visual datasets with QCNN - Image processing
  - Improvement in low-resource language classification with QRNN - NLP
  - Potential improvements in scheduling tasks with QNN - Supply chain management
  - Secure encryption even against quantum attacks - Cybersecurity
- Expectations are for quantum-classical hybrid computing

doitnow
HPC Services




Credit: Amazon Braket

doitnow HPC Services

# Terminology braindump

| | |
|---|---|
| QPCA | Quantum Principal Component Analysis |
| QRNN | Quantum Recurrent Neural Networks |
| QRL | Quantum Reinforcement Learning |
| QSVM | Quantum Support Vector Machine |
| QTSA | Quantum Time-Series Analysis |

| | |
|---|---|
| AI | Artificial Intelligence |
| CNN | Convolutional Neural Networks |
| DL | Deep Learning |
| EQDA | Exploratory Quantum Data Analysis |
| GAN | Generative Adversarial Networks |

| | |
|---|---|
| QLSTM | Quantum Long Short-Term Memory |
| QML | Quantum Machine Learning |
| QNLP | Quantum Natural Language Processing |
| QNB | Quantum Naive Bayes |
| QNN | Quantum Neural Networks |

| | |
|---|---|
| TFQ | TensorFlow Quantum |
| VQA | Variational Quantum Algorithm |
| VQC | Variational Quantum Classifiers |
| VQE | Variational Quantum Eigen |

| | |
|---|---|
| QIML | Quantum Inspired Machine Learning |
| QIP | Quantum Image Processing |
| QKD | Quantum Key Distribution |
| QKNN | Quantum K-Nearest Neighbour |

| | |
|---|---|
| NISQ | Noisy Intermediate-Scale Quantum |
| NLP | Natural Language Processing |
| PCA | Principal Component Analysis |
| PQC | Post-Quantum Cryptography |
| QAOA | Quantum Approximate Optimization Algorithm |

| | |
|---|---|
| QTL | Quantum Transfer Learning |
| RL | Reinforcement Learning |
| RNN | Recurrent Neural Networks |
| SVM | Support Vector Machine |

| | |
|---|---|
| QC | Quantum Computing |
| QDA | Quantum Data Analysis |
| QEC | Quantum Error Correction |
| QFL | Quantum Federated Learning |
| QGAN | Quantum Generative Adversarial Networks |

| | |
|---|---|
| GMM | Gaussian Mixture Models |
| HPC | High Performance Computing |
| KNN | K-Nearest Neighbour |
| LSTM | Long Short-Term Memory |
| ML | Machine Learning |

doitnow HPC Services

# 3 building blocks for QML

1 ENCODE (Quantum)

- Feature map

2 ALGORITHM (Quantum)

- QNN, quantum kernel, etc.

3 PROCESSING (Classical)

- Cost function, optimizer, etc.

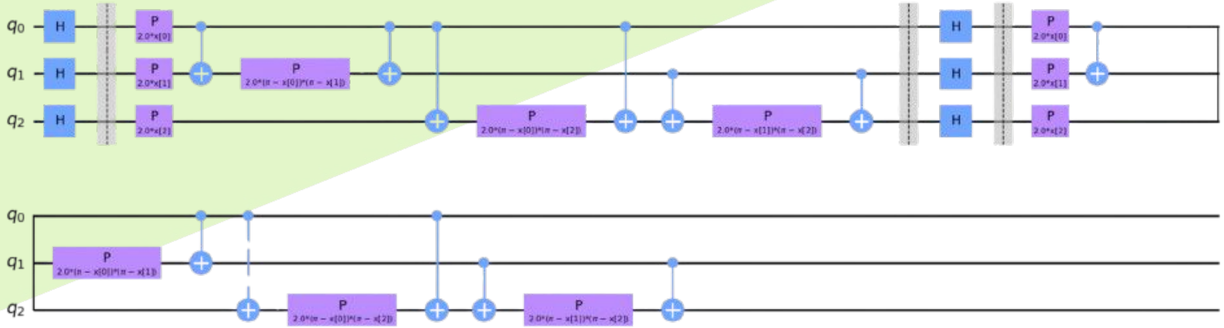
doitnow
HPC Services

# Feature map (fm)

$$\{x_i, y_i, z_i\} \longrightarrow \boxed{\text{Feature Map}} \longrightarrow |\psi_i\rangle$$

- Classical data → Quantum states
- Lower dimension → Higher dimension (Hilbert feature space)
- #features & encoding method determines #qubits
- #repetitions affect smoothness:
  - More reps
  - More gates over each feature
  - More volatility when feature changes
  - Less smoothness
- Popular examples:
  - ZFeatureMap
  - ZZFeatureMap ⟶

(3 qubits with 2 repetitions)



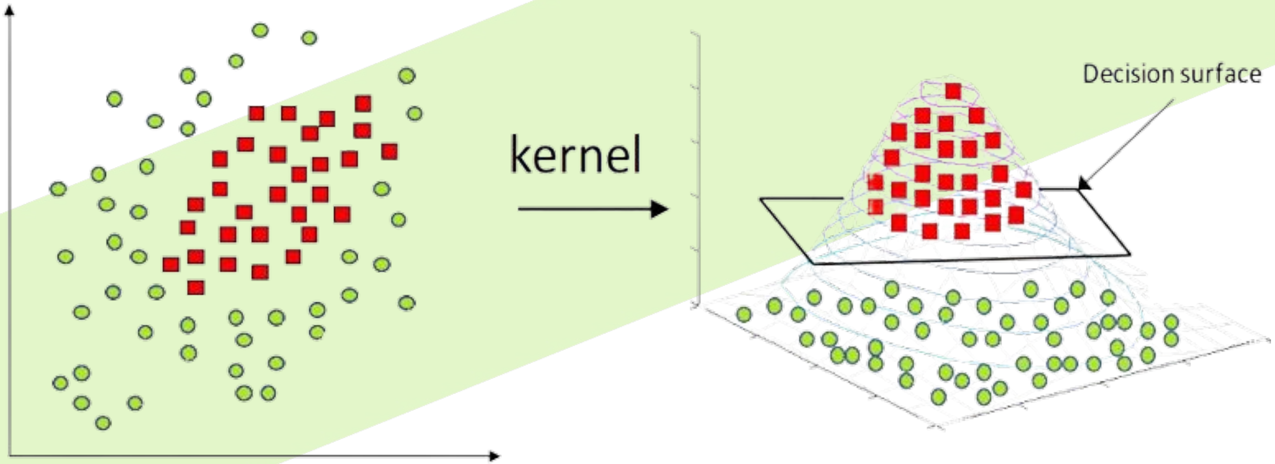
# Encoding method

- Basis encoding
  - Bit string representation → Computational state representation
- Angle encoding
  - $n$ features → $n$ qubits rotated around an axis e.g. Z-axis
- Dense angle encoding
  - Same, but with 2 features per qubit on 2 axes e.g. Z-axis & X-axis
- Amplitude encoding
  - $n$ features → Probability amplitudes of $n$ qubits i.e. $0 \longrightarrow \underset{|\beta|^2}{\longrightarrow} \overset{|\alpha|^2}{\longrightarrow} \longrightarrow \longrightarrow 1$

Entanglement key to quantum improvement opportunity!

# Support Vector Classifiers (SVC)

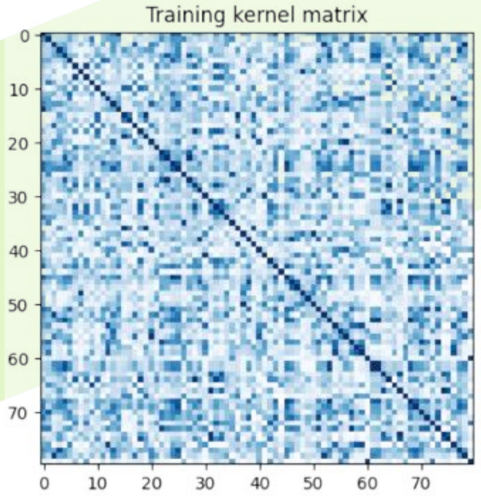
- Transfer points to higher dimension ( ← quantum naturally does! )
- Trained hyperplane classifies data points
- Hyperplane is the lowest cost line, farthest from data points per class
- Kernel function does the trick

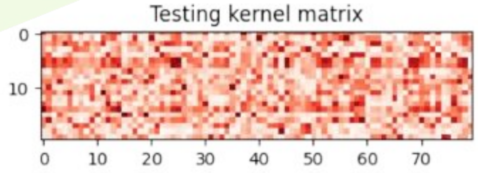


doitnow HPC Services

# Kernels

- Kernel function can be represented by a matrix
- Classical kernel matrix: $K_c = [K_{ij}\ \forall i,j] = [\langle f(\vec{x_i}), f(\vec{x_j})\rangle\ \forall i,j]$ ; $f$ is map
- Quantum kernel matrix: $K_q = [K_{ij}\ \forall i,j] = \left[|\langle\phi(\vec{x_i})|\phi(\vec{x_j})\rangle|^2\ \forall i,j\right]$ ; $\phi$ is feature map
    - This is the fidelity of (pure) states
    - Quantum Kernel Alignment (QKA): can train feature map to optimize quantum data pre-kernel

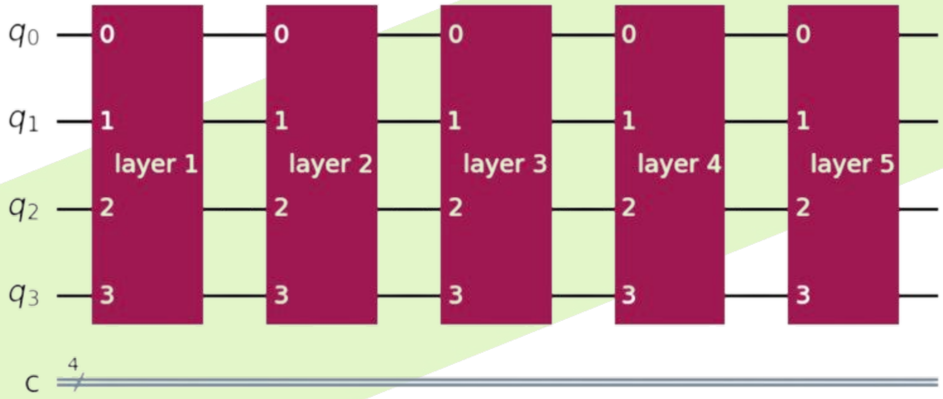


doitnow HPC Services

# Ansatz

- Parameterized circuit trainable weights
- Each layer of ansatz acts as layer of neural network
- More weights → More fine-tuning
- #features & #layers determines #parameters
- Popular examples: RealAmplitudes, EfficientSU2

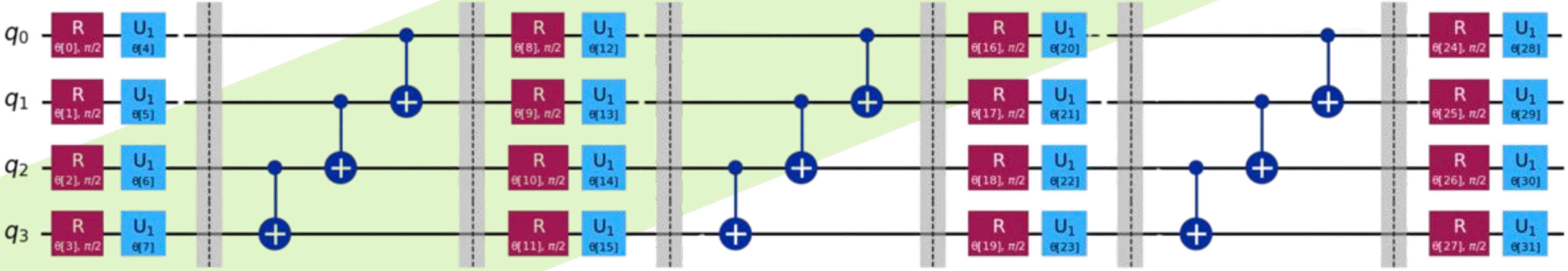



Layers can be repetitions of same smaller circuit

## Ansatz

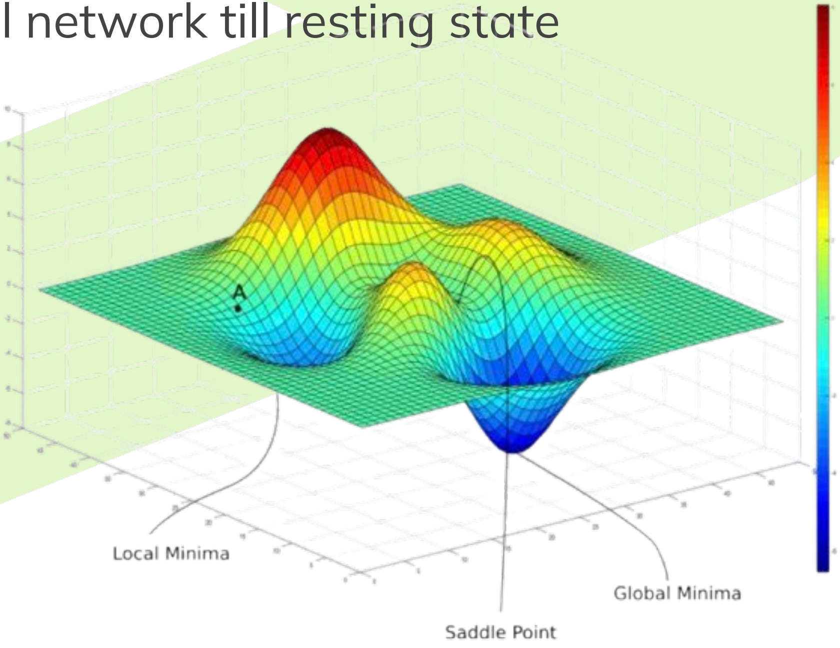
- Optimizer after the ansatz minimizes the cost function, fine-tunes weights
- Again!... entanglement encouraged to exploit quantum advantages
- Can't be too much depth currently due to noise
- Notice the 'pre-layer' gates → Priming and avoiding loss of first-layer utility

Barriers indicate 3 layers



# Optimizer

- Classic method to find local or global minima cost
- Iterates with new parameters for neural network till resting state
- Examples with ideal usage:
  - SPSA: Gradient descent
  - COBYLA: Objective functions with unknown derivatives
  - SLSQP: Twice continuously differentiable objective functions
  - ADAM: Large and noisy datasets



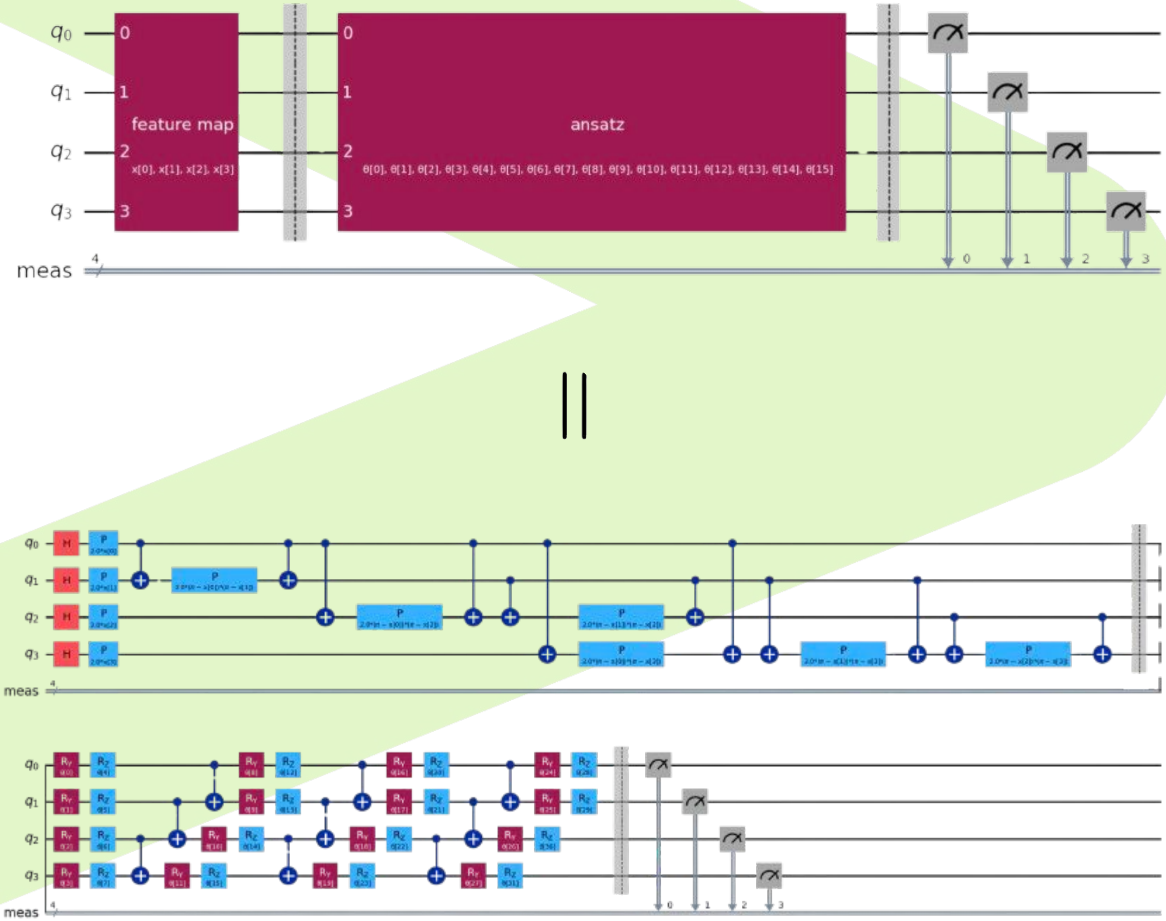
# Variational Quantum Classifier (VQC)

- Supervised neural network classifier as an example of VQAs (..... Algorithms)
- Introduced by [Havlíček et al. in 2019]
- Interpret expectation values as classifier output
- Promising for NISQ era (Noisy Intermediate-Scale Quantum)
- Often requires Principal Component Analysis (PCA) dimensionality reduction
- Requires many measurements to get expectation values
- NOTE: VQC can also mean Variational Quantum Circuits, be wary...

$$
\begin{tikzcd}
data_{cl} \arrow[r] & fmap() \arrow[r, mapsto] & data \arrow[r] & U_{data}(\theta_i) \arrow[r] & meas() \arrow[r, mapsto] & labels \arrow[r, Rightarrow] \arrow[dr] & model \\
 & & \theta_1 \arrow[ur] & \theta_{i+1} \arrow[u] & opt() \arrow[l] & cost \arrow[l] & C() \arrow[l]
\end{tikzcd}
$$

# VQC example

- Feature map: 1 rep
  - ZZFeatureMap
- Ansatz: 3 reps
  - RealAmplitudes
- Measurements: 1000
  - Get expectation values
- Cost function: pain
  - MSE
- Optimizer: 100 epochs
  - COBYLA
- Dataset: Iris



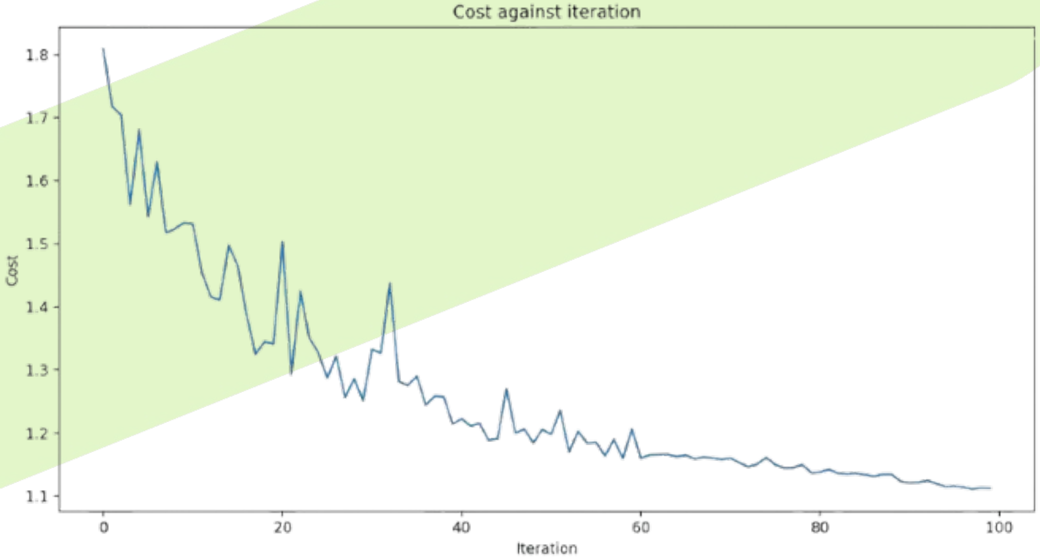
# VQC example

```python
import time
from matplotlib import pyplot as plt
from IPython.display import clear_output
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris
from qiskit_algorithms.optimizers import COBYLA
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit.visualization import plot_histogram

# Get, scale and split dataset
iris = load_iris()
X = iris.data
y = iris.target
X = MinMaxScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=123)

# Define core components for VQC
feature_map = ZZFeatureMap(feature_dimension=X.shape[1], reps=1)
ansatz = RealAmplitudes(num_qubits=X.shape[1], reps=3)
optimizer = COBYLA(maxiter=100)

# Define callback graph for visualizing cost minimization
plt.rcParams["figure.figsize"] = (12, 6)
def callback_graph(weights, obj_eval):
    clear_output(wait=True)
    vals.append(obj_eval)
    plt.title("Cost against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Cost")
    plt.plot(range(len(vals)), vals)
    plt.show()

# Fit data to the VQC
vqc = VQC(feature_map=feature_map, ansatz=ansatz, optimizer=optimizer, callback=callback_graph)
vals = []
start = time.time()
vqc.fit(X_train, y_train)
elapsed = time.time() - start

# Get results
print(f"Training time: {round(elapsed)} seconds")
train_score_quantum = vqc.score(X_train, y_train)
test_score_quantum = vqc.score(X_test, y_test)
print(f"VQC on the training dataset: {train_score_quantum:.2f}")
print(f"VQC on the test dataset:     {test_score_quantum:.2f}")
```

Training time on laptop: 27 seconds (Macbook M2)

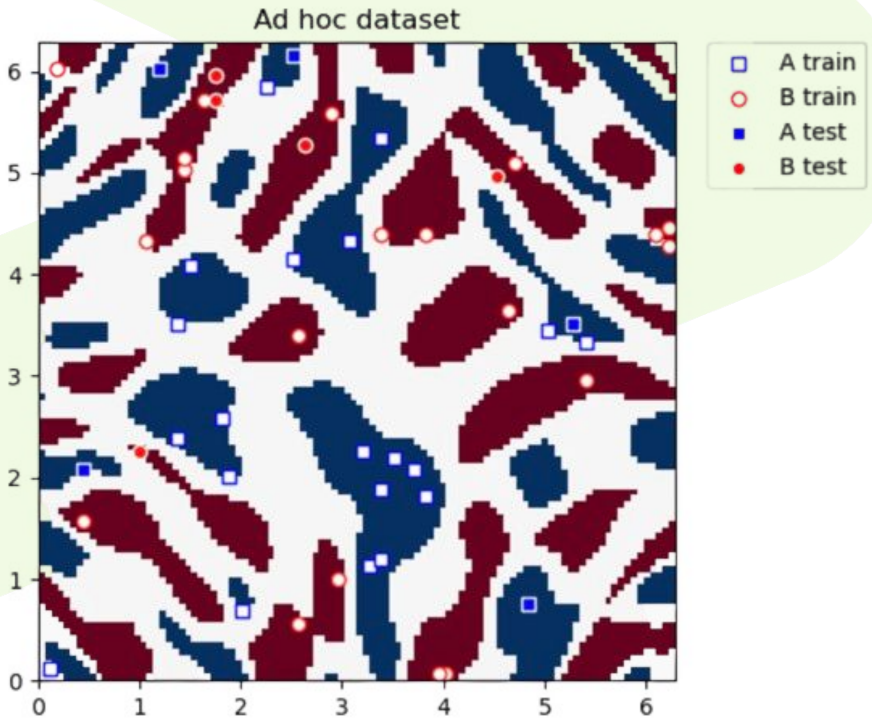
VQC on training data: 0.79

VQC on testing data: 0.73



# Havlíček et al. paper drawbacks

- The dataset is “perfect” for quantum.
- Results were 100% classification, but real datasets show weaker score than their classical counterpart.
- Datasets like Iris with 4 features and 3 classes, QSVC and VQC previously did at best classical-level accuracy.
- Same went for MNIST.



doitnow
HPC Services

# VQC these days…

- VQC and QSVC for prediction in vaccine design (2025)
  - VQC slightly outperforms QSVC and SVC, which were about equal
- Parallelized VQC with pre-determined shallow QRAM (2024)
  - Much better accuracy for Iris and MNIST due to log reduction in circuit width (more dim. allowed)
- VQC dimensionality reduction with autoencoders (2024)
  - QAE (quantum autoencoder) outperforms AE and PCA on Iris, MNIST and other
- High-efficiency VQC for high-dimensional data (2024)
  - No dimensionality reduction required, with improved results

People love their pretty Iris flowers

# Benchmarking ← Always a good idea

Which feature map?
- ZFeatureMap
- ZZFeatureMap
- Other
- Custom
- AI gen.?

How many repetitions/layers?
- 1,2,3,4,5...

Which ansatz?
- RealAmplitudes
- EfficientSU2
- Other
- Custom
- AI gen.?

Credit: Abigail Burrola

Which optimizer?
- SPSA
- COBYLA
- SLSQP
- ADAM

How much dim. reduction?

Which dim. reduction?
- PCA
- Autoencoder
- Other

Which entanglement structure?
- linear, circular, full, custom...

doitnow
HPC Services

# Now to different neural networks