❮ WOMANIUM | QUANTUM ❯

Qml for conspicuity detection

BY SHREYAS PATIL & RONIT DUTTA

Overview

The project is divided into five tasks. The first task was to complete codebooks in Pennylane, thus familiarizing ourselves with the workflow and Pennylane library. Tasks 2 and 3 introduced us to variational classifiers and QNN architectures, working us through the QML pipeline. Task 4 involved building our own QML model to approximate the sine function. Task 5

overview

- TASK 1
- TASK 2
- TASK 3
- TASK 4

Task 1

SECTION 1

Familiarize with the basic principles of quantum computing mainly concepts of linear algebra, complex numbers, states, inner/outer products, unitaries etc.

SECTION 2

Familiarize with single qubit operation, identities, custom unitaries, rotations, universal gates, expectation values etc.

SECTION 3

Multi qubit operations, entanglement, controlled unitaries etc

Task 2

PART 1 : THE PARITY PROGRAM

- This tutorial intends to build a classifier for the IRIS dataset.
- We have referred to the parity routine for help with the same. To do so we build a pqc as an ansatz with tunable rotation gates for optimal parameters of the ansatz state.
- The cost function is given by the square loss function of labels and predictions.
- With the functions given the ansatz parameters are optimized via a gradient descent algorithm and the corresponding accuracy of the predictions are calculated as bool values.

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}(\text{Prediction}(i) - \text{Label}(i))^2$$

$$\text{Acc} = \frac{1}{n}\sum_{i=1}^{n}\text{Bool}(\text{label}(i) - \text{prediction}(i) = 0)$$

Part 2: Flower classification of the IRIS Dataset

- **Target**

  Classify the different types of flowers in the IRIS dataset.

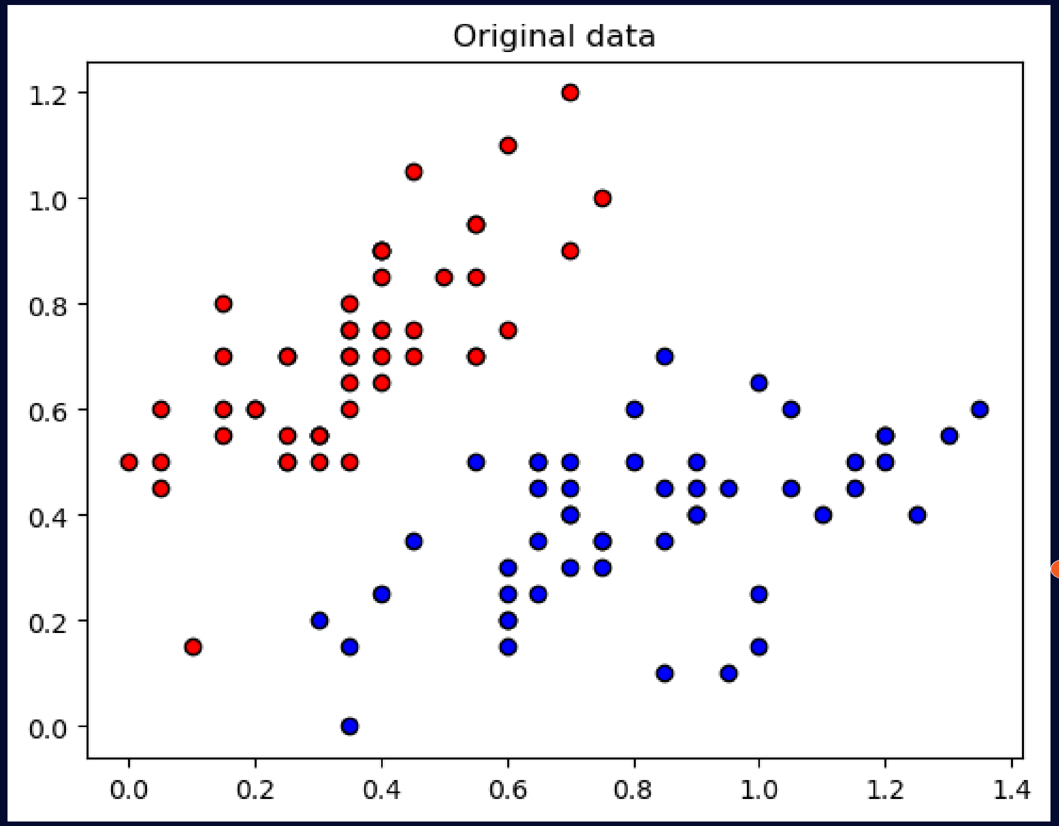
- **Method**

  Unlike the Parity Function, the Iris dataset's continuous variables require encoding into angular rotations, which are then represented as amplitudes via RY rotations and entangled with CNOT gates. Simple basis state preparation isn't sufficient here.

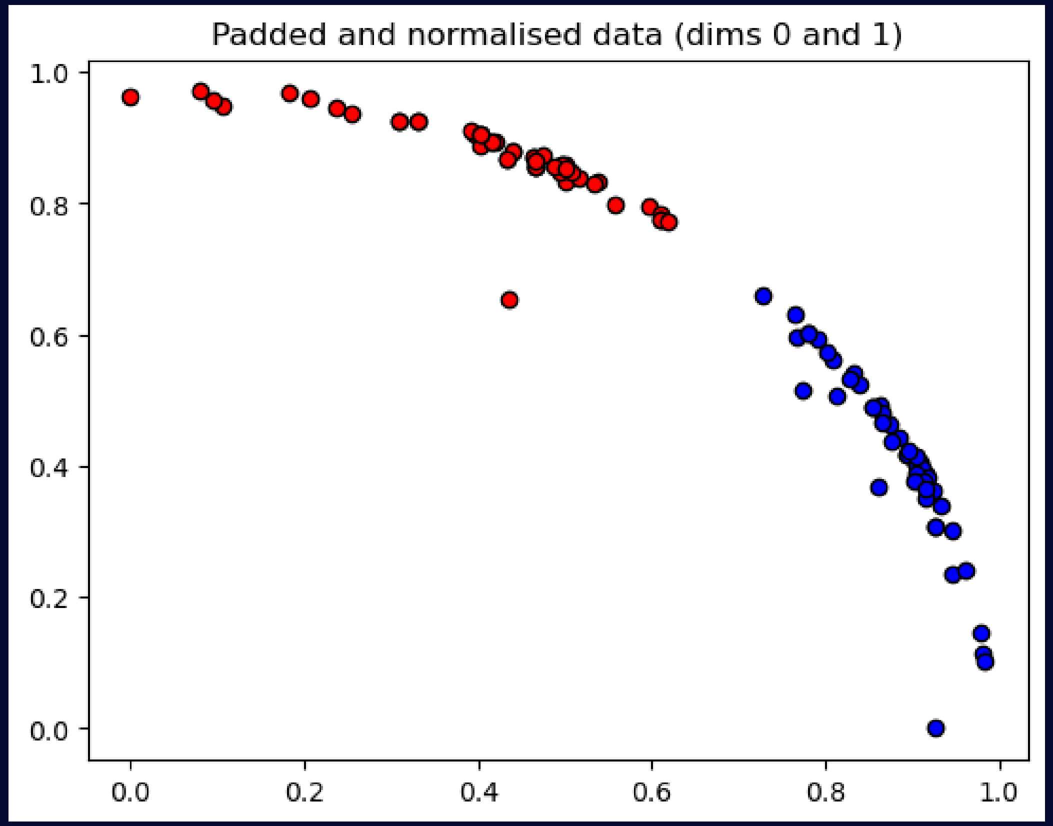
- **Output**

  The data is preprocessed and to make it suitable for quantum computing we apply suitable amplitude encoding. Then it’s split into test train sets.

WOMANIUM | QUANTUM

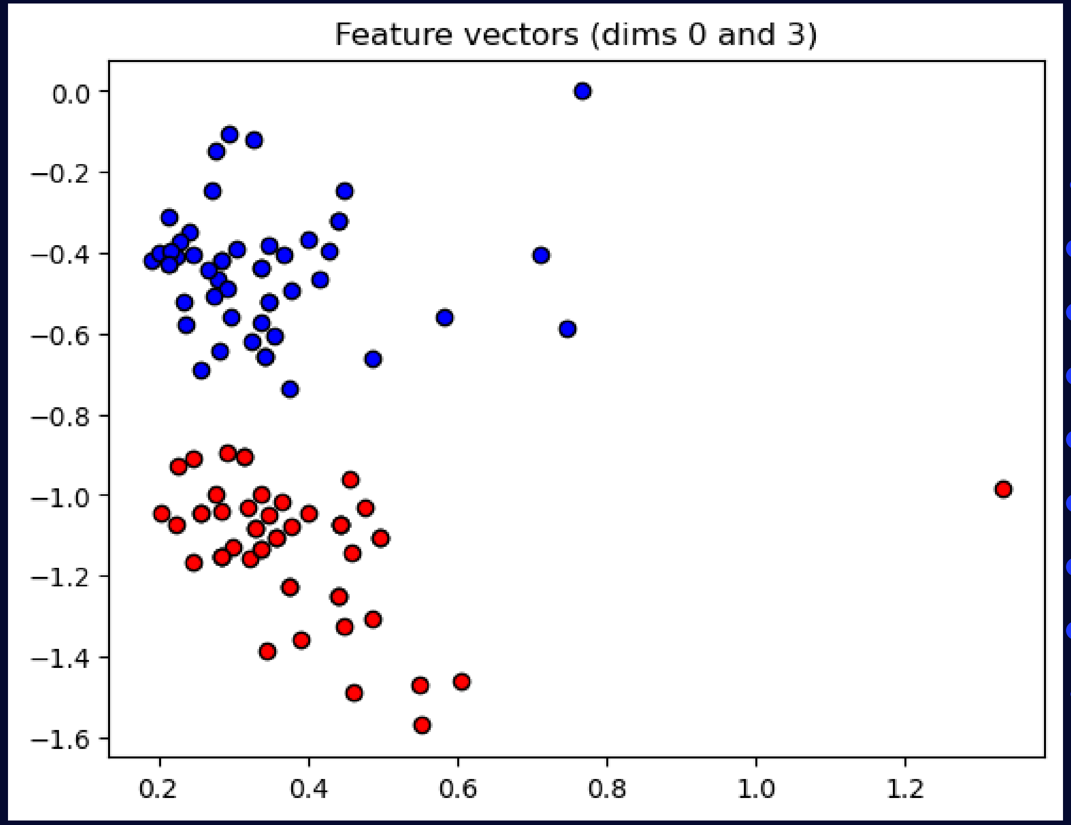


Original data

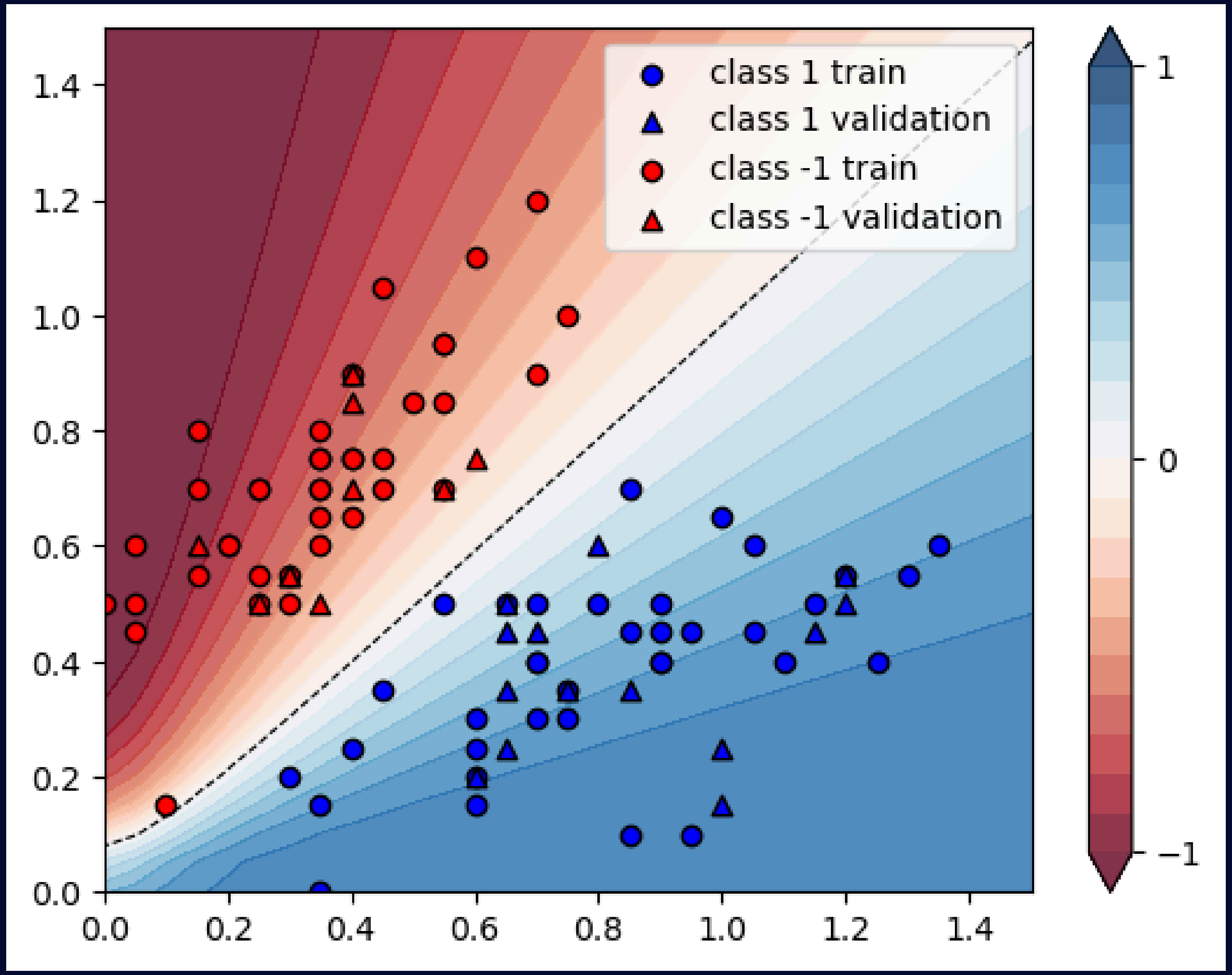


Padded and normalized
to match the amp encoding
dimension

feature vectors which will be
used as basis state.
amplitude encoding routine of qml.



Result



- After optimization of the circuit ansatz parameters and following a routine closely similar to the parity function we are able to classify between the two flowers and the results are shown here.
- The hyperplane is drawn as well.
- We get excellent validation accuracy for the same.

TASK 3

Intro : Quanvolutional Filters

- Quanvolutional filters are designed to produce feature maps by applying quantum transformations to spatially-local subsections of an input tensor.
- Unlike classical convolutional filters that rely on simple element-wise matrix multiplication, ***quanvolutional filters*** utilize quantum circuits to transform the input data.
- These quantum circuits can be either structured or randomly generated, providing a unique approach to feature extraction in quantum neural networks.

Our mission and accomplishment

● OBJECTIVE 1

Use the MNIST Handwritten Digits dataset to identify the numbers and test for accuracy using a quanvolutional neural network.

● OBJECTIVE 2

Show quantum supremacy by comparing results of the qcnn with that of the classical cnn results.

Preprocess the data and prepare features

- **Target**

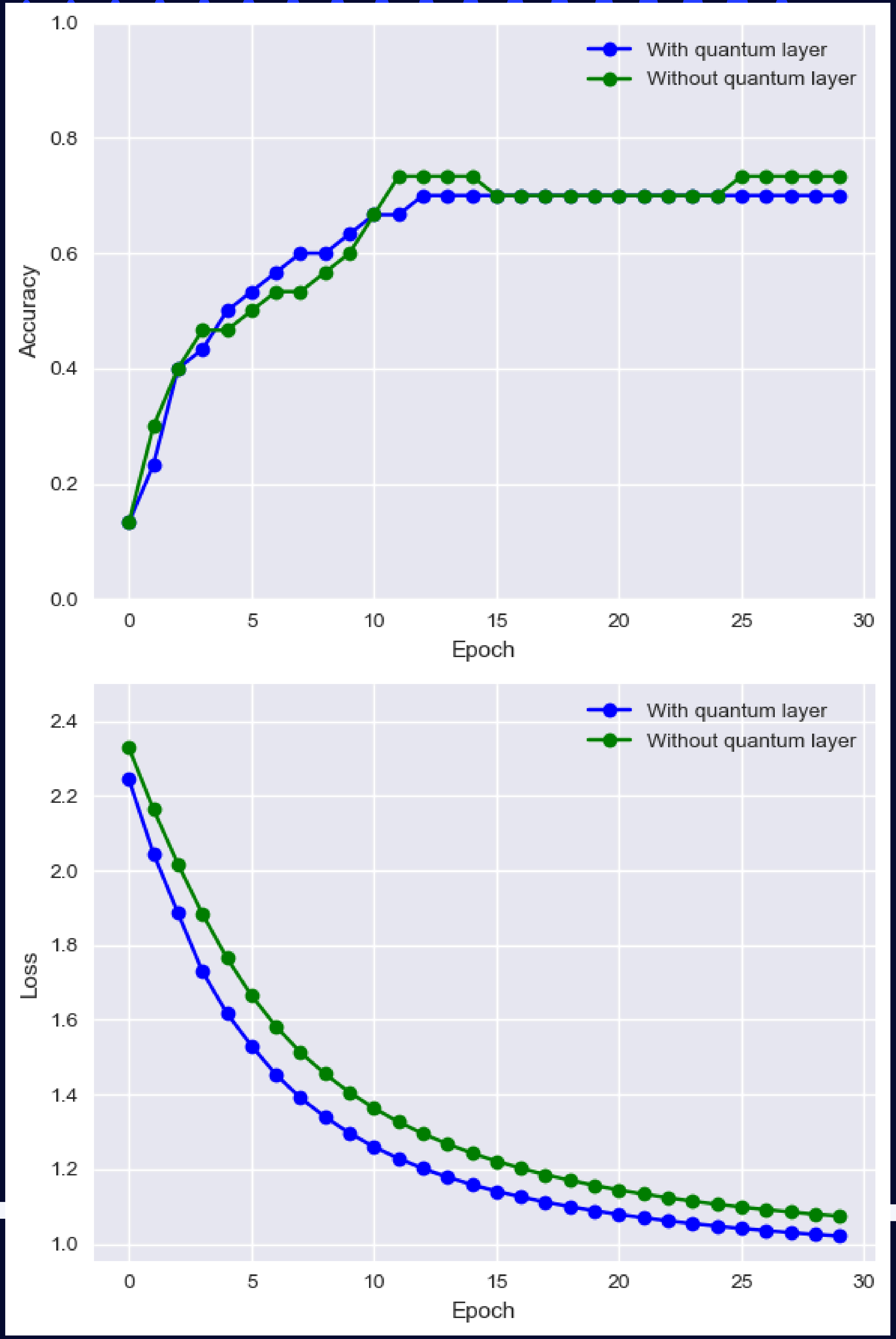
  Preprocess the data by applying the quanvolutional layer.

- **Method**
  1. After splitting into testing and training data, preprocess the images using the quanvolutional layer for the pixel map of the images.
  2. Use keras function library to pool the features from the convolutional layer, train a quantum and classical model separately and ultimately generate feature vectors.

- **Accuracy calculation and quantum classical comparison**

  Calculate the respective accuracy of the features and compare the same with it's classical counterpart and plot results.

Result



The classical and quantum accuracy and loss trend has been plotted together to give a clear and visual idea of how the both models compare among themselves.

Task 4

BASIC TASK

The objective of this subtask is to develop a quantum model capable of learning the sine function over the interval $[0,2\pi]$.

INITIAL STEP

We approached this by discretizing the interval into a suitable number of points and using the corresponding sine values at these points as labels.

COMPARISON

We build another classical neural network for fitting the sine function and then compare fitting results and prediction accuracy between the quantum and the classical neural network model.

Quantum Implementation

- **Basic Idea**
  The idea here is to use a parametrized circuit whose expectation value will be the approximation of the sine function.
- **Step 1**
  In our solution, we use a single qubit for the circuit. We encode the value of x into the circuit by performing an RY operation with the value of x, since x is also the angle and lies in the range of $[0,2\pi]$.
- **Step 2**
  We define a layer to contain a single parametrized rotation operator. This is the optimization parameter consisting of psi, theta, omega which shall be optimized. The resulting pauliZ expectation value is our Cost function.

```
0: ──RY(0.00)──Rot(0.00,0.00,0.00)─┤  <Z>
```

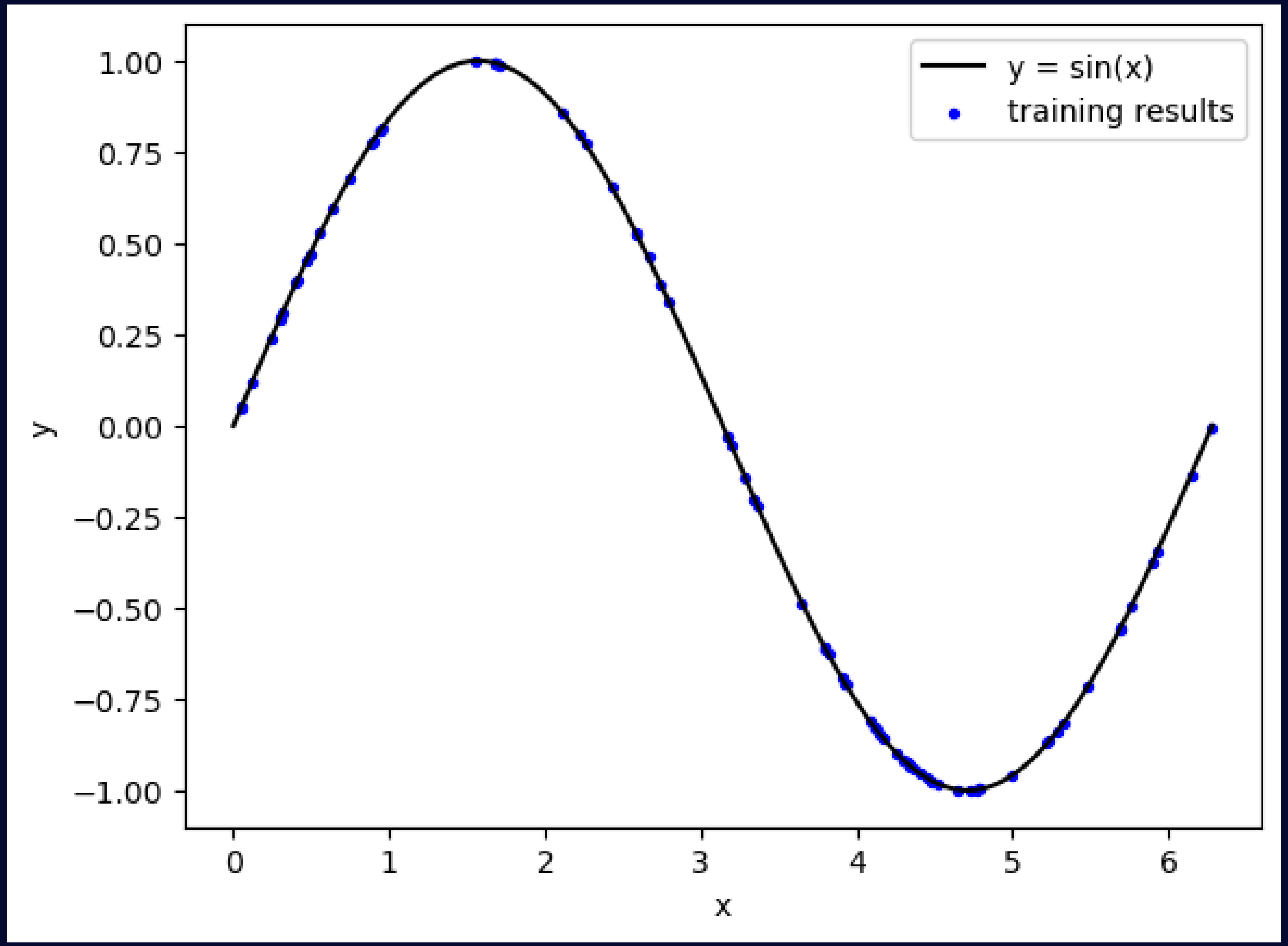
Optimization circuit consisting of only 1 qubit

1.Create Datasets

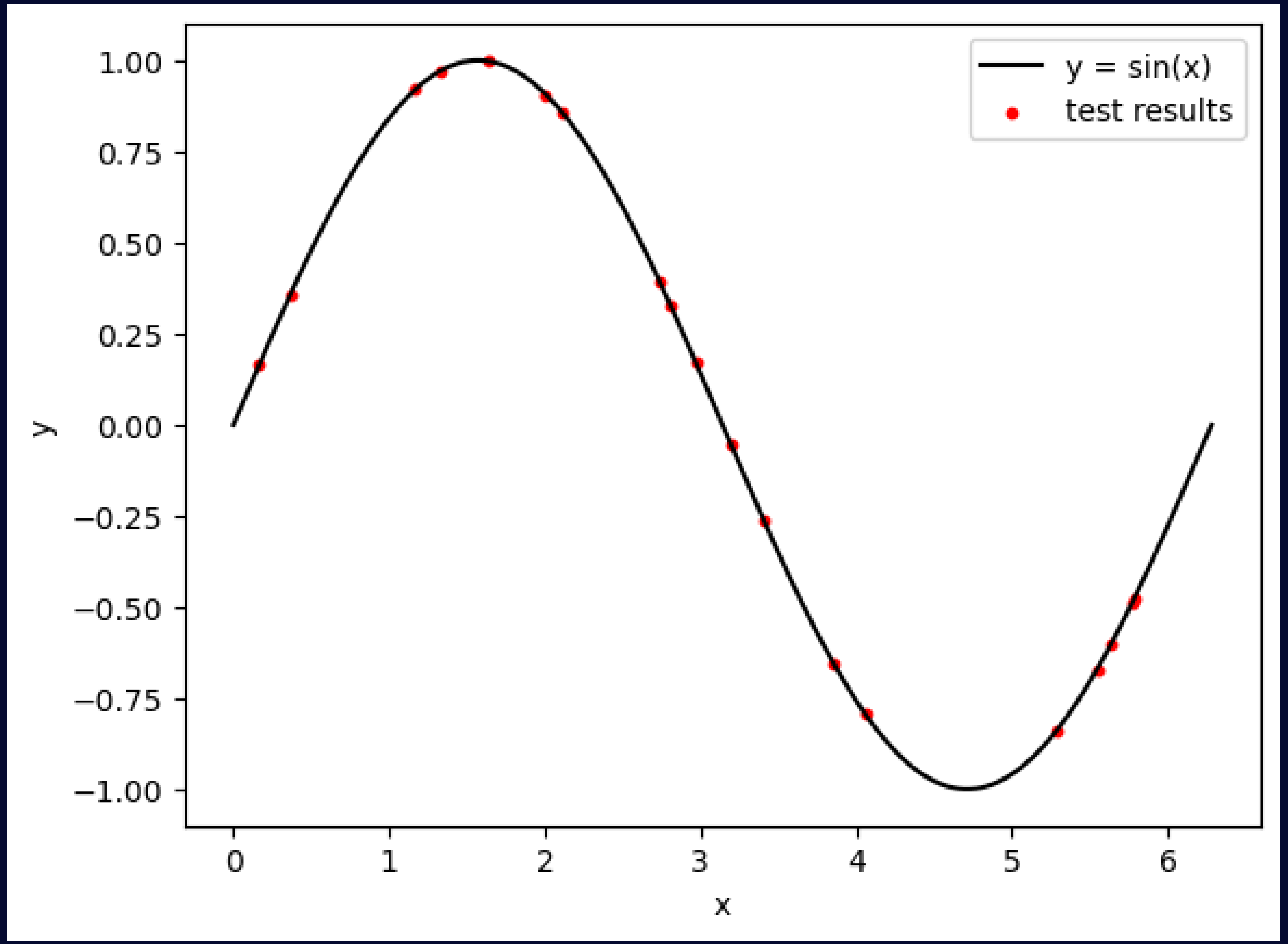
We discretized the interval [0, 2π] into 10,000 points for continuity and accuracy. From these, we randomly sampled 100 points (without loss of generality) to create our train and test datasets, ensuring an 80:20 split between training and testing.

2.Optimization

- We propose an ansatz of num_layers = 1, and randomly initialize weights and biases close to zero.
- We use the Nesterov Momentum Optimizer, with a step size of 0.3.
- We set the batch size to 16 since that's neither a lot nor too less, for performing Stochastic Gradient Descent on a total of 100 samples and train for 40 epochs



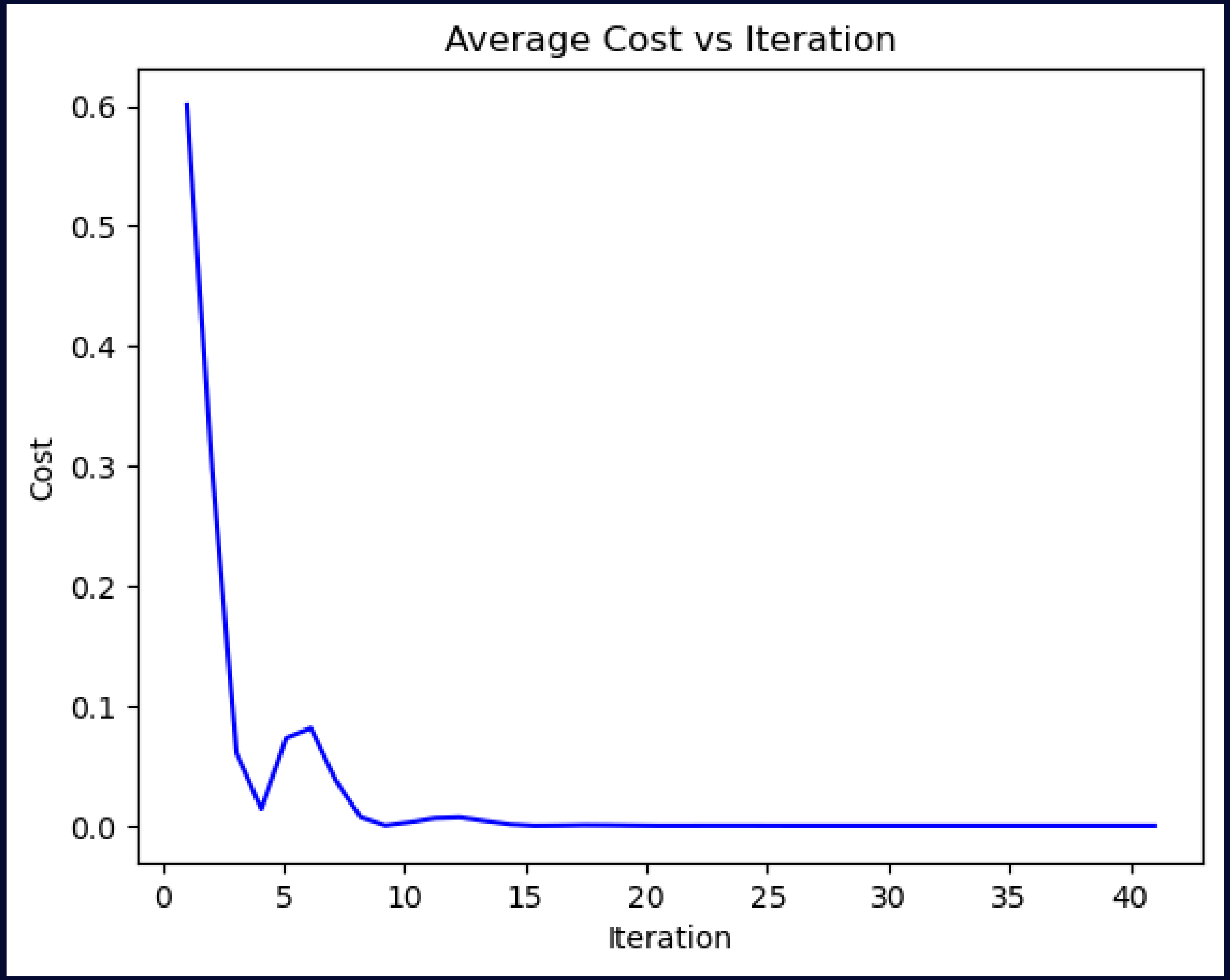
Training data (left) and testing data (right)



Training data

Testing data

Trend of cost function



Classical implementation

- **Method**

  We implement a classical Neural Network made up of three hidden layers to approximate the sine function. Discretize the interval of $[0,2\pi]$ into a thousand points. Distribute them radomly into training and test datasets in 80:20 ratio. We use the test dataset to validate our trained model, to make sure it hasn't under/overfit the training data.

- **Next step**

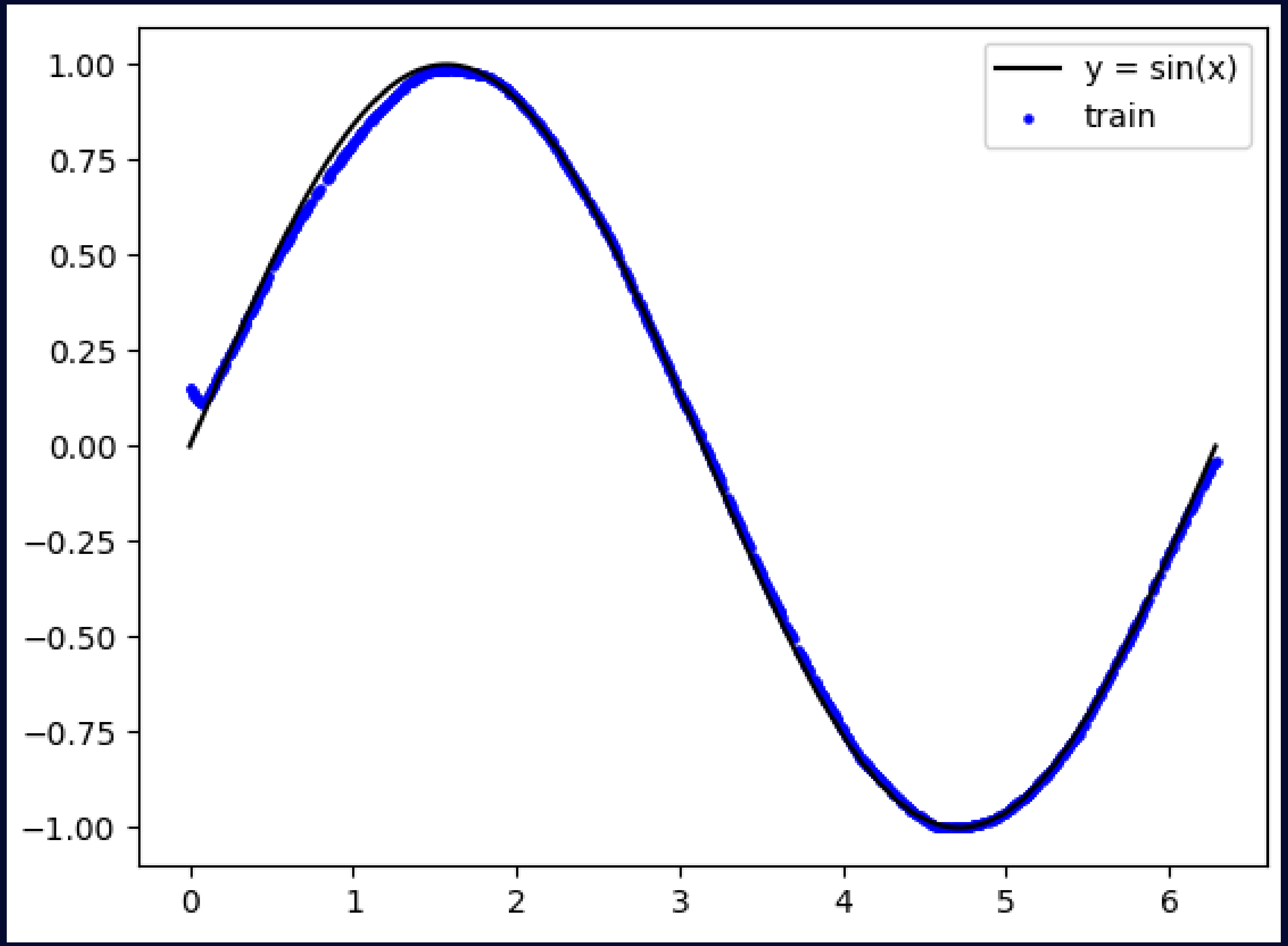
  Each layer is succeeded by a ReLU activation function. This provides the non-linearity to the network. The layer after is the output.

- **Next step**

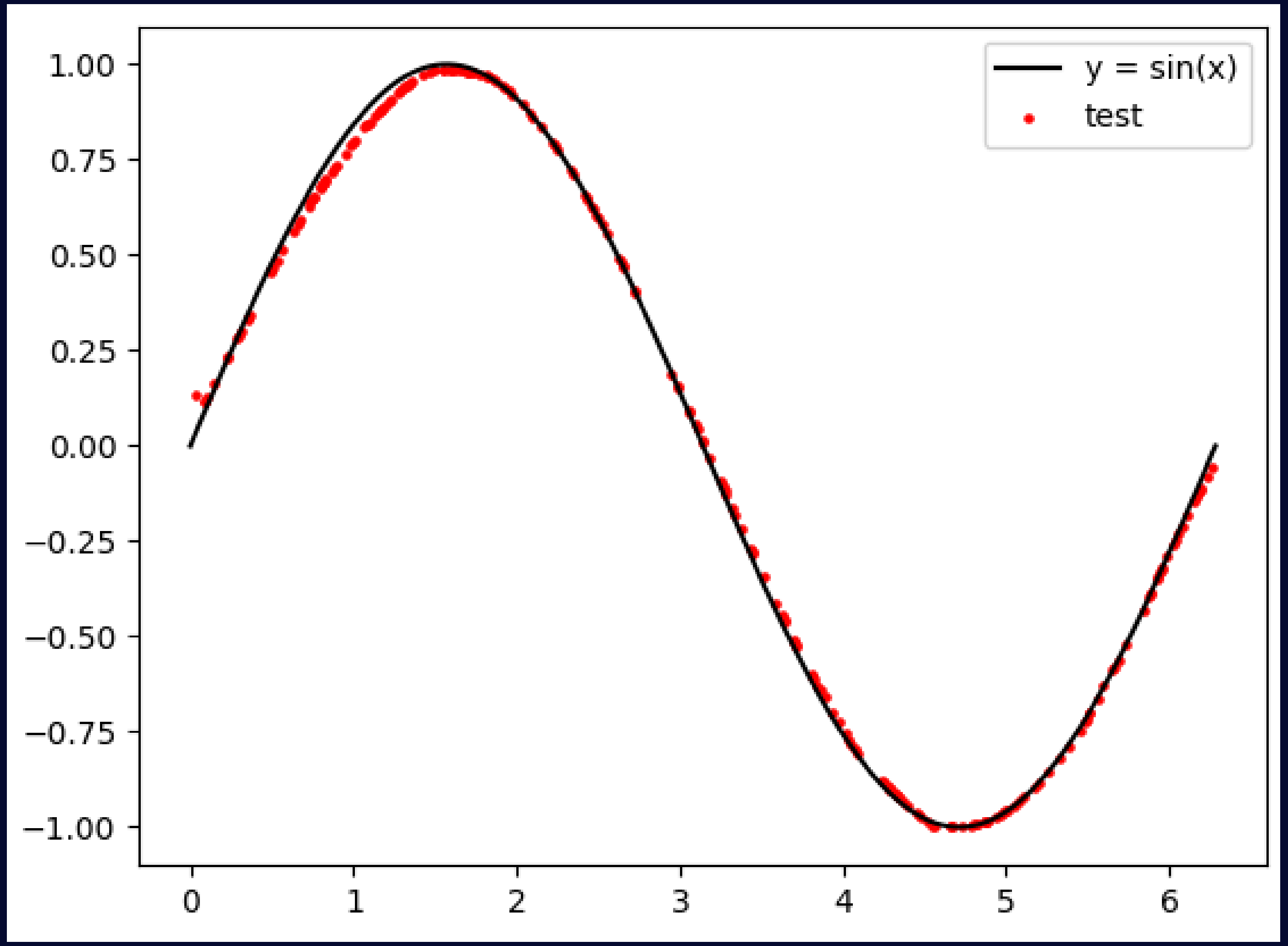
  We use square loss for our loss function. We also use a scheduler that drops the Learning Rate by a factor at a defined number of steps so that the gdo algorithm doesn’t encounter a local minima problem. The optmization routine is run for 100 epochs next.

WOMANIUM | QUANTUM

Classical implementation



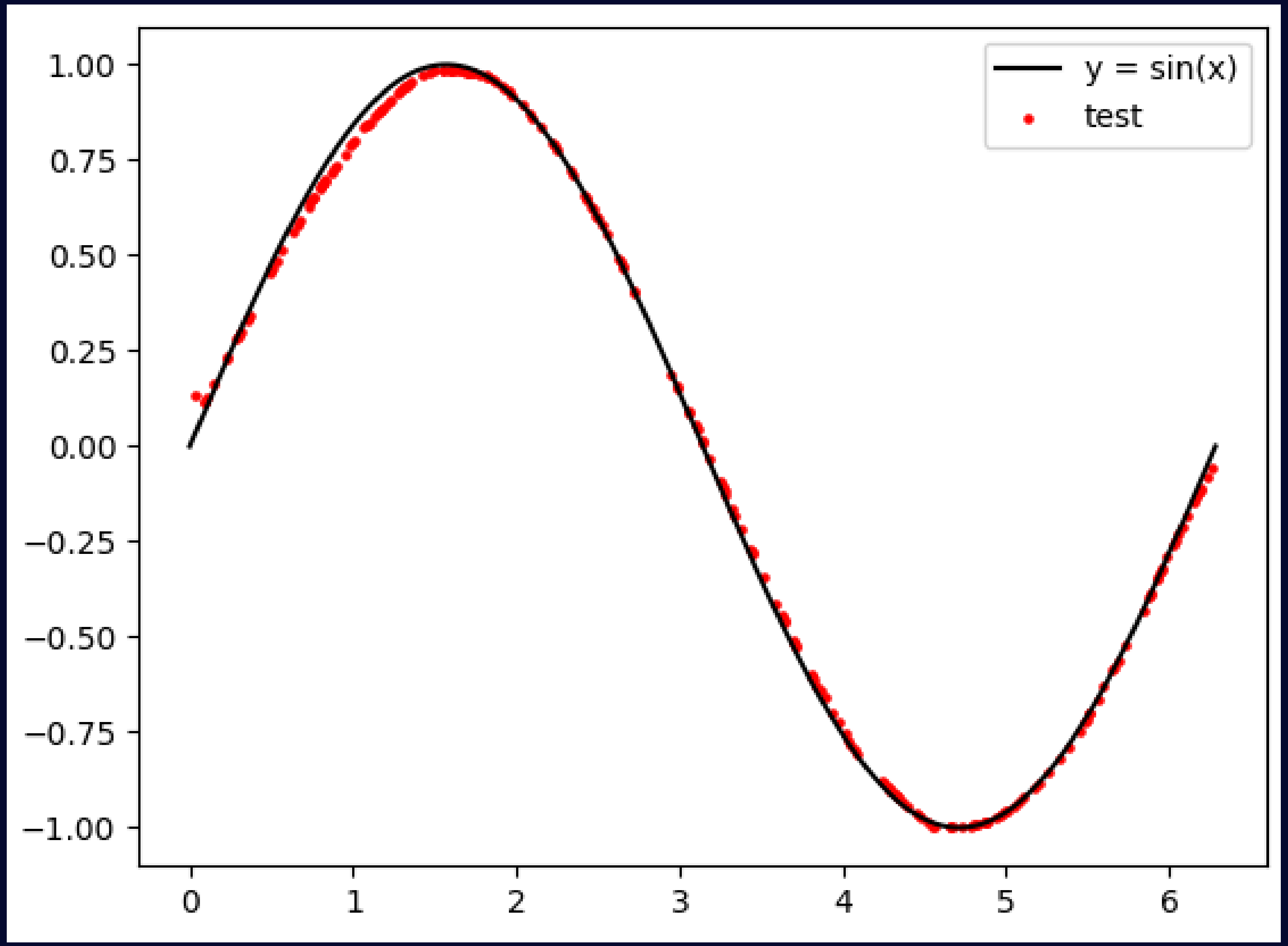
Training data (left) and testing data (right)



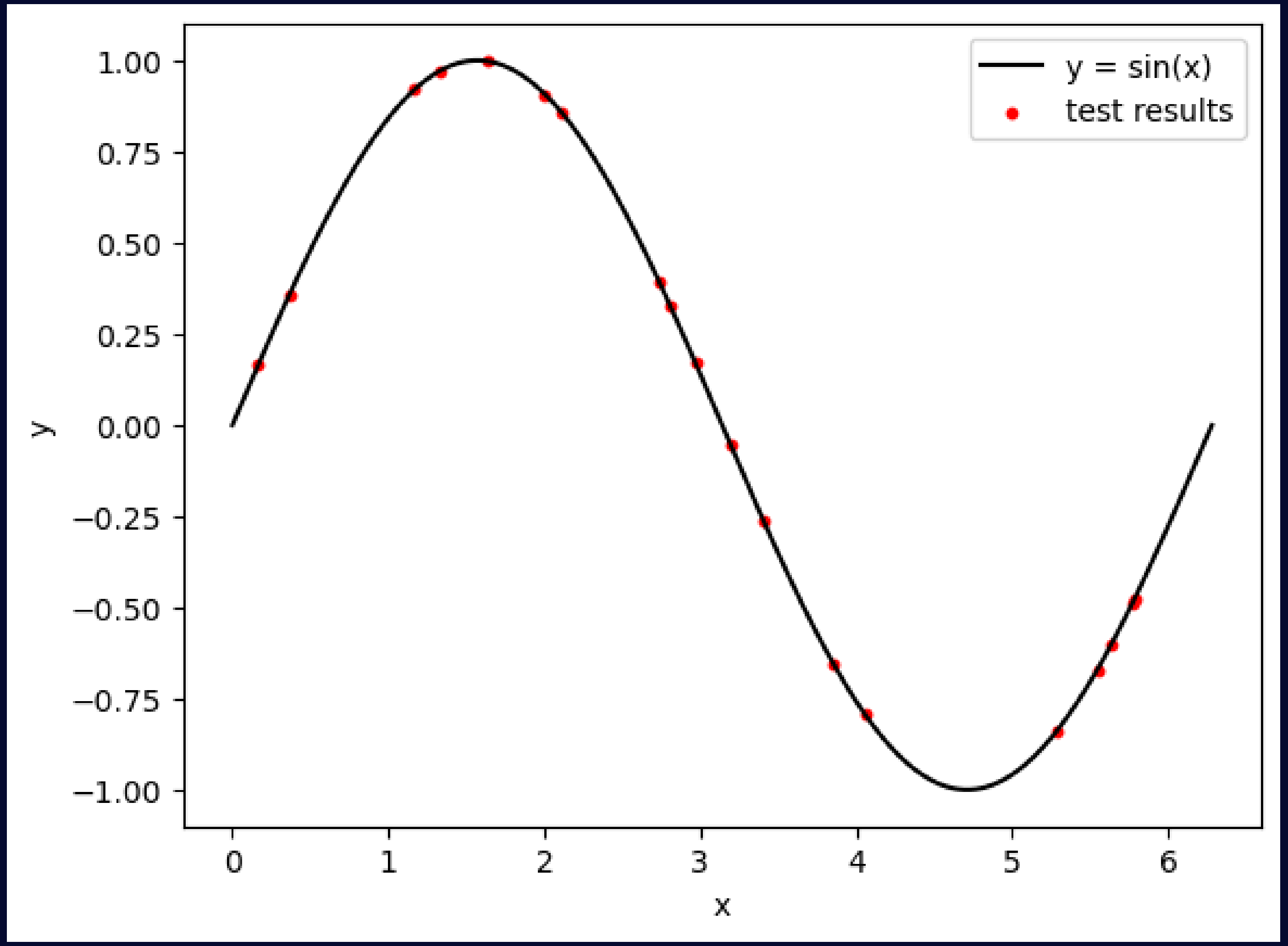
Training data

Testing data

Comparison



Classical Function fitting



Quantum Function fitting

WOMANIUM QUANTUM+AI PROJECT

‹ WOMANIUM | QUANTUM ›

Thank You

by Shreyas Patil and Ronit Dutta