International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Training Doubly Fed Induction Generator Performance Parameters Using Variational Quantum Regressor and Classifier Models

Mamidi Ramakrishna Rao

Head Research Engineer, DHI–QUEST Private Ltd, Hyderabad, India

Abstract

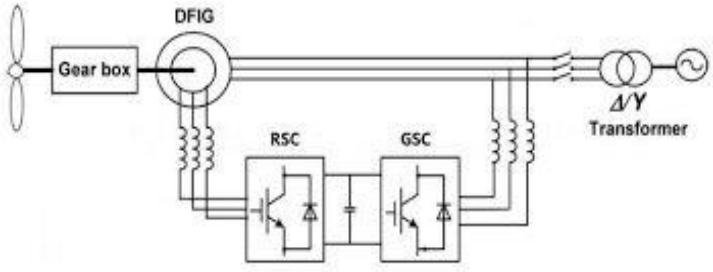
Renewable energy systems, particularly wind energy conversion systems, are increasingly adopting advanced data-driven techniques for performance modelling and control. Among wind generators, the doubly fed induction generator (DFIG) is widely used due to its high efficiency, variable-speed operation, and four-quadrant capability. In this work, Variational Quantum Machine Learning (QML) models are employed to analyse key DFIG performance parameters. The rotor current and reactive power are selected as target outputs and modelled using a Variational Quantum Classifier (VQC) and Variational Quantum Regressor (VQR), respectively. Four critical DFIG design and operating features are encoded into quantum feature space using Qiskit-based feature maps. The quantum models are trained on a dataset of 1000 specially developed DFIG designs spanning a power range of 1000 kW to 2100 kW. Model performance is evaluated in terms of prediction accuracy and convergence behaviour, highlighting the feasibility and potential advantages of variational quantum algorithms for power system component modelling in the near-term quantum computing era.

Keywords: DFIG, Quantum Machine Learning, QISKIT

1. Introduction

A DFIG-based wind energy conversion system consists of several major subsystems, including the wind turbine with blades, gearbox, doubly fed induction generator, and power electronic converters—namely, the rotor-side converter (RSC) and the grid-side converter (GSC)—along with their associated control systems [1],[2]. Each of these subsystems play a significant role in determining the overall dynamic and steady-state performance of the system.

Figure. 1 Schematic diagram of a DFIG wind turbine



In accordance with the objectives of this study, a subset of key parameters that have a dominant influence on system performance is selected for detailed analysis.

1.1. Wind Speed and DFIG Shaft Speed [3]

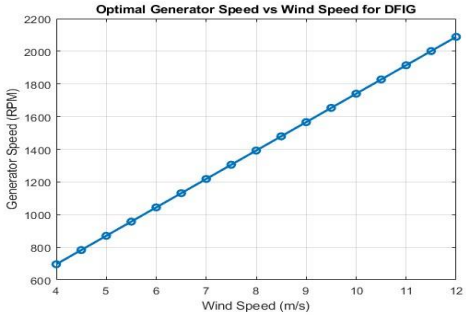
The rotational speed (rpm) of the DFIG and the mechanical power input are directly related to the wind speed (m/s). This relationship is governed by several aerodynamic and mechanical factors, including the gearbox ratio, which couples the low-speed turbine shaft to the high-speed generator shaft, the turbine blade radius, blade pitch angle, air density, power coefficient (Cp), torque coefficient (Ct), and tip-speed ratio (λ\lambda).

These parameters collectively determine the aerodynamic efficiency, torque production, and power output of the wind turbine, enabling efficient operation over a wide range of wind speeds.

DFIG is characterized by a wound-rotor induction machine whose rotor circuit is connected to the grid through a back-to-back power electronic converter. This configuration enables variable-speed operation and independent control of active and reactive power, while the power electronic converter typically processes only 20%–30% of the rated generator power.

For a specific operating condition, the relationship between wind speed and generator speed is presented in figure 2.

Figure 2: The relation of wind speed and Generator speed.



1.2 Reactive Power and Voltage Support Capability [4]

In a DFIG-based wind energy conversion system, reactive power exchange with the grid is a primary control variable for meeting grid-code compliance. The DFIG control structure enables **decoupled control of active and reactive power at the point of common coupling (PCC)**, making reactive power particularly suitable for regulation and optimization.

According to modern grid codes, wind power plants are required to provide **dynamic reactive power and voltage support**, particularly during grid disturbances and voltage deviations. By controlling the reactive power injected into or absorbed from the grid, the DFIG can effectively regulate the **PCC voltage magnitude**, thereby contributing to voltage stability and improved power quality.

From an efficiency perspective, optimal allocation of reactive power between the stator and the grid-side converter (GSC) reduces **stator copper loss, rotor copper loss, and converter conduction and switching loss** across the wind farm. This coordinated reactive power control improves the overall efficiency of the wind energy conversion system.

The vector control strategy applied to the DFIG enables independent regulation of:

Active power, typically controlled through the rotor-side converter (RSC) for maximum power point tracking (MPPT), and

Reactive power, controlled to satisfy grid voltage or power factor requirements at the PCC .
This inherent decoupling allows reactive power regulation without compromising active power extraction under normal operating conditions

1.3 Rotor Current and Converter Protection

Rotor current is a critical variable during transient operating conditions, particularly under grid fault scenarios. The rotor-side converter (RSC) is typically rated at **25%–30% of the generator rated power**, and excessive rotor currents during voltage dips may result in overcurrent stress and potential damage to the power electronic devices.
Grid codes require DFIGs to remain connected during grid disturbances, which necessitates strict rotor current limitation strategies. Effective rotor current control directly enhances the **dynamic performance of the system**, reducing overshoot and improving settling time following sudden wind speed changes or grid voltage variations, thereby ensuring smooth active power recovery.

1.4 Fault Ride-Through (FRT) and Low-Voltage Ride-Through (LVRT) compliance [4]

Modern grid codes mandate **fault ride-through (FRT)** capability for wind turbines, requiring continued operation during short-duration voltage depressions at the PCC. In particular, **low-voltage ride-through (LVRT)** requirements specify voltage–time profiles that DFIGs must withstand without disconnecting from the grid.
During LVRT events, rotor current suppression and controlled reactive current injection are essential to:

- Prevent overcurrent in the RSC,
- Maintain DC-link voltage stability, and
- Provide voltage support to the grid in accordance with grid-code requirements.

Optimized rotor current control is therefore a key factor in ensuring grid-code compliance during fault conditions.

1.5 Secondary Performance Metrics

Certain parameters, while relevant, are considered secondary in the context of grid-code–oriented control optimization. System weight is primarily a mechanical design constraint and does not directly affect dynamic grid interaction. Rotor voltage is an important internal control variable; however, it is typically a derived quantity determined by rotor current and converter control actions.
System efficiency remains a key steady-state performance metric, as it reflects the effectiveness of aerodynamic energy conversion and electrical power transfer under normal grid conditions

2 Case Studies Formulated

Based on grid-code compliance requirements and control objectives, two case studies are defined:
Case I: Rotor current is selected as the target variable, with generator speed, stator current, reactive kVA and rotor voltage used as input features.
Case II: Reactive power at the point of common coupling (PCC) is selected as the target variable, with rotor current, rotor voltage, generator speed, and active power as input features

3. Doubly Fed Induction Generator (DFIG) Dataset Preparation

The dataset is prepared for DFIGs with rated power levels ranging from **1000 kW to 2100 kW**, which correspond to typical utility-scale wind turbine applications. The machine designs are generated through detailed **electromagnetic design and analysis**, ensuring physical feasibility and compliance with practical operating constraints.

A total of **nine independent machine design variables [5]** are selected, as listed in Tables I and II. These variables serve as the primary inputs for the electromagnetic design process and are sufficient to derive all additional dependent parameters required for comprehensive performance evaluation, including flux distributions, losses, torque, and efficiency.

The lower and upper bounds of each design variable are determined based on **manufacturing constraints, material limitations, thermal considerations, and standard industrial practices**. This ensures that all generated designs are realizable and suitable for practical implementation [6].

For the **2100 kW DFIG**, the corresponding design variable ranges and calculated electromagnetic parameters are presented in **Tables I and II**. Similar datasets are generated for **1000 kW and 1500 kW** rated machines using the same design methodology and constraint framework to maintain consistency across power ratings.

In total, **1000 distinct DFIG design instances** are generated to form the complete dataset used for subsequent analysis and model training.

Figure 3: Variables used for design data generation

a) Magnetic core variables **b) Stator and rotor slot variables**

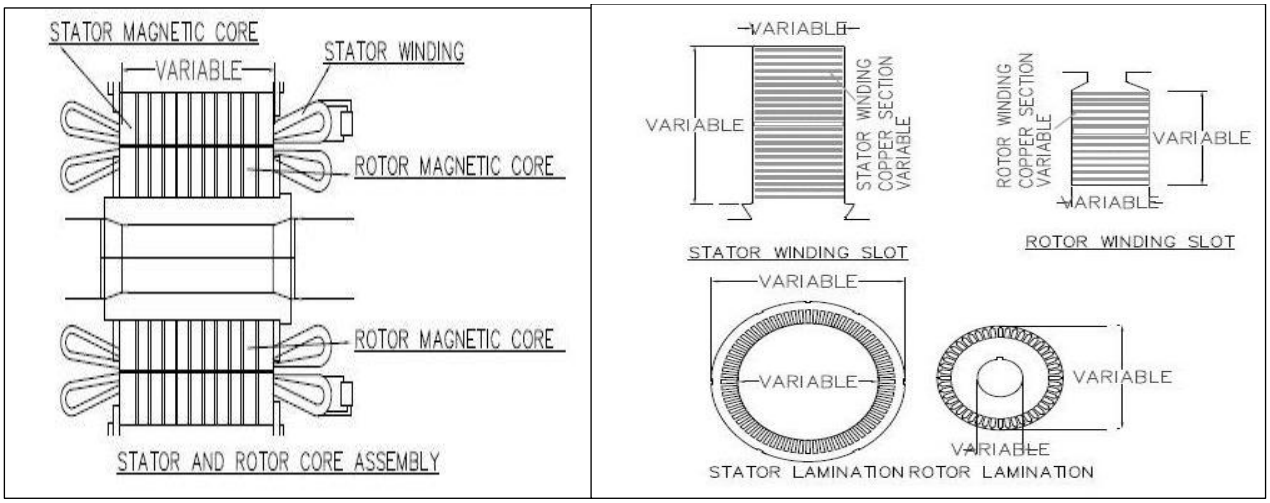


Table 1: Rated Electrical and Mechanical Parameters of the 2100 kW DFIG [1]

Parameter	Value
Rated stator voltage (Volts)	690 ± 10%
Rated grid frequency (Hz)	50
Number of poles	4
Synchronous speed (rpm)	1500
Maximum operating speed (rpm)	1900
Minimum operating speed (rpm)	1000
Rated electromagnetic torque (Nm)	13,504
Power factor capability	0.95 (capacitive) / 0.95 (inductive)

Table 2: Design Variable Limits for the 2100 kW DFIG [1]

Variable	Limits (mm)
Rotor inner diameter (D2)	390.0 < D2 < 500.0
Rotor core depth (CD2)	90.0 < CD 2 < 120.0
Rotor minimum teeth width (TW2)	15 .0< TW2 <25.0
Rotor teeth height (TH2)	55.0< TH2 <70.0
Radial air gap (AG)	2.5 < AG < 5.0
Stator minimum teeth width (TW1)	15.0 <TW1 <25.0
Stator teeth height (TH1)	45.0 < TH 1 < 60.0
Stator core depth (CD1)	90.0 < CD 1 <130.0
Stack length (CL)	850.0 < CL < 1000.0

4. Variational Quantum Machine learning (Training) [8]

For training the **Variational Quantum Regressor (VQR)** and **Variational Quantum Classifier (VQC)** models, the learning problem is formulated by defining the **labels**, **targets**, and **quantum feature space** as follows.

Labels (Class Encoding): The machine ratings of **1000 kW, 1500 kW, and 2100 kW** are treated as discrete class labels and encoded for the VQC model.

Targets (Regression Outputs):

- **Case 1:** Rotor current
- **Case 2:** Reactive kVA

Input Features and Quantum Feature Map: The classical input variables are embedded into a quantum Hilbert space using a parameterized **feature map**, where each feature corresponds to one input dimension of the quantum circuit.

- **Case 1 Feature Vector:** x1=[DFIG speed, stator current, rotor voltage, reactive kVA]
- **Case 2 Feature Vector:** x2=[DFIG speed, stator current, rotor voltage, rotor current]

Each feature vector has an **input dimension of four**, and is encoded into the quantum state via angle encoding, resulting in a **four-qubit quantum circuit**.

Although additional electrical and mechanical parameters could be incorporated to enrich the feature space, increasing the input dimension directly impacts **qubit scaling** and circuit depth. To balance model expressiveness with hardware and simulation constraints, the number of input features is therefore restricted to four in each case, ensuring manageable qubit requirements and reduced variational circuit complexity.

4.1 Encoding Classical Data into Quantum States [10]

Classical data must be mapped into quantum states before being processed by variation quantum algorithms. Several data encoding techniques have been proposed in the literature, including **basis encoding**, **amplitude encoding**, **angle encoding**, **phase encoding**, and **dense angle encoding**.

In **Qiskit**, classical-to-quantum data embedding is typically achieved using **quantum feature maps**, which project classical input vectors into a high-dimensional **quantum feature space (Hilbert space)**. This representation enables quantum circuits to capture complex correlations that may be difficult to model classically.

The commonly used feature maps available in Qiskit include:

- **EfficientSU2 feature map**

- **Z Feature Map** (PauliFeatureMap without entanglement)
- **ZZ Feature Map** (PauliFeatureMap with entanglement)
- **Pauli Feature Map**

In this work, a **second-order Pauli-Z evolution–based feature map** is employed. The corresponding quantum circuit is constructed using **repeating layers**, each comprising the following components:

- **Hadamard Layer:** An initial layer of Hadamard gates is applied to all qubits to prepare an equal superposition state, ensuring that the encoded data explores the full Hilbert space.
- **Single-Qubit Rotation Layer:** Parameterized single-qubit rotation gates are applied to each qubit, where the rotation angles are directly determined by the individual components of the classical input feature vector. This step encodes the input data into the quantum state using angle-based encoding.
- **Entanglement Layer:** Controlled two-qubit Pauli-Z (ZZ) rotation gates are applied between selected pairs of qubits using either **linear** or **full entanglement** topologies. The rotation angles of these entangling gates are governed by nonlinear functions of the input features, enabling the circuit to model higher-order feature correlations.
- The number of qubits required scales linearly with the **input dimension**, while the depth of the circuit increases with the order of feature interactions and the number of repetition layers. Consequently, feature map selection plays a critical role in balancing **expressibility**, **qubit scalability**, and **circuit depth**, especially under **NISQ-era hardware constraints**.
- It is important to note that **no single feature map is universally optimal** for all datasets. The effectiveness of a given feature map depends on the structure, dimensionality, and correlation patterns present in the data. Therefore, the choice of feature map must be guided by both the problem characteristics and practical quantum resource limitations.
- In Qiskit, the most widely used classical optimizers for variational quantum algorithms include **L-BFGS-B** and **COBYLA**, along with other optimizers such as **SLSQP** and **SPSA**. The **L-BFGS-B** optimizer is a gradient-based quasi-Newton method. Although it can achieve high precision, it requires greater computational effort per iteration, exhibits relatively slower convergence, and is more susceptible to becoming trapped in local minima.
- In contrast, **COBYLA (Constrained Optimization BY Linear Approximations)** is a gradient-free optimization technique, making it more suitable for noisy or gradient-inaccessible quantum circuits. In the present **DFIG performance analysis**, the **COBYLA optimizer** is employed for both the **Variational Quantum Classifier (VQC)** and the **Variational Quantum Regressor (VQR)** models.
- The training was performed for **100 iterations**, and the corresponding training time and optimized variational parameters are summarized below.

4.2 Variational Quantum Classifier (VQC) : [6]

It is a hybrid quantum – classical algorithm to classify data. It encodes data into quantum states, applies trainable gates (ansatz) measures the output and uses a classical optimizer to update parameters, minimizing a loss function. The core components of VQC are Data Encoding, Ansatz (Variational Circuit), Measurement and Mapping, and classical Optimizer.

The ZZ feature map and Z feature map , both are used to encode classical data into quantum state by utilizing both individual qubit rotations. The key difference between the ZZ feature map and simple Z_feature_map lies in the **inclusion of entangling gates**.

In this work for for VQC analysis, (‘Ir2’ Target) the configuration of “Z feature map” is shown in the figure 4 for VQC analysis.

Figure 4. Configuration of Z Feature Map used in VQC

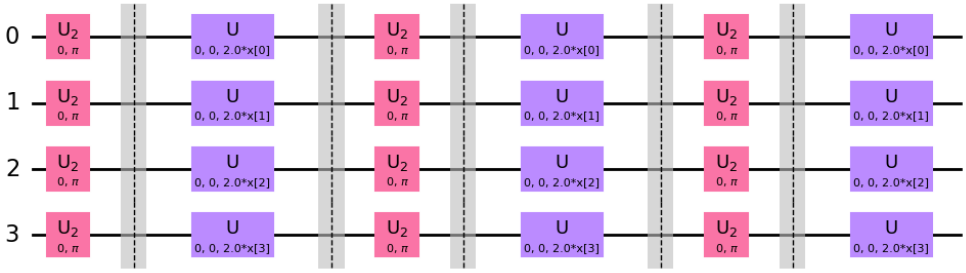
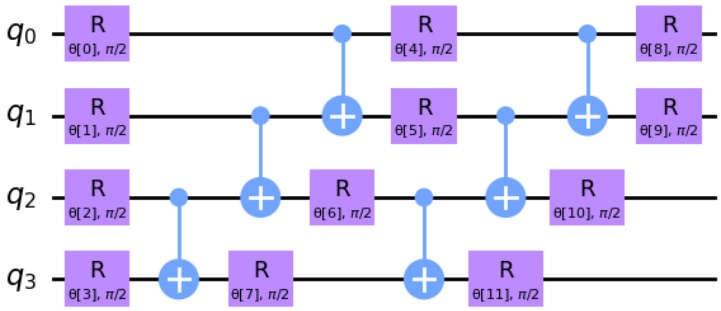


Figure 5. Configuration of Ansatz in VQC



4.2.1 VQC Results:

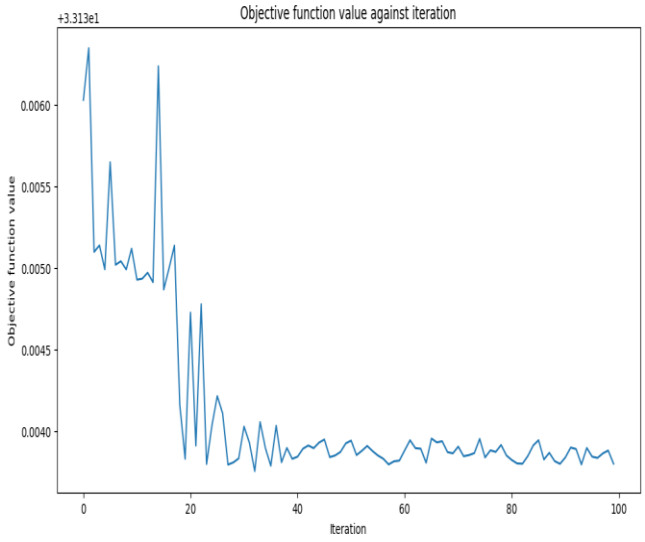


Figure 6a : Iteration vs Objective function values (with ZZ feature map)

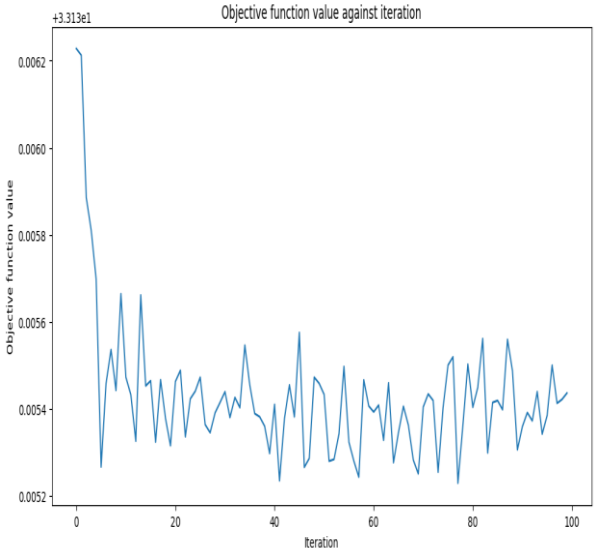


Figure 6b : Iteration vs Objective function values (with Z feature map)

Training time: 930 seconds (with ZZ feature map)

Final Optimized Angles: [33.135446239223484, 33.135976506684045, 33.13559339027544, 33.13536930487619, 33.135442272220004, 33.135346395641044, 33.1354854654892, 33.13543447410799, 33.135258147248386, 33.13561959015717, 33.13531562832763, 33.13545524430063, 33.13524675747174, 33.13540743649792, 33.13533937816808, 33.13529350459762, 33.13528003303964, 33.13537435127248, 33.13523614108189, 33.135847210667755, 33.1353039271128, 33.13537777566709, 33.135301777378636, 33.1354470202388, 33.13516740523958, 33.13538655136191, 33.13526813137589, 33.13542826915469, 33.13542371519084, 33.13548756249773, 33.13552267584095, 33.135462806656676, 33.13538767554026, 33.135468682849684, 33.13532448716996, 33.13551118569677, 33.13542847541104, 33.13536762099758, 33.13541288603387, 33.13533840188312]

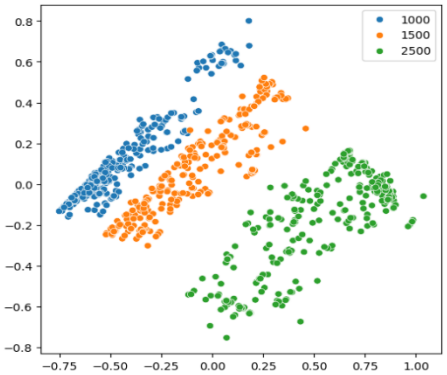


Figure 7a : Scatter plot of features using Min Max scalar

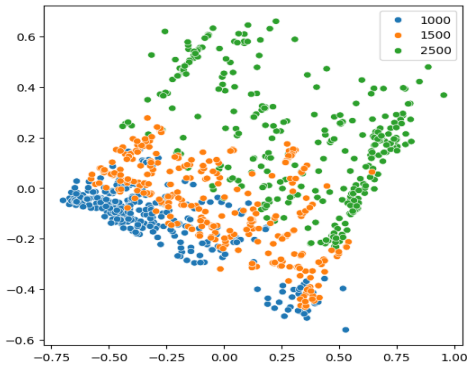


Figure 7b : Scatter plot of features using Standard scalar

4.3 Variational Quantum Classifier (VQR) [7]:

Variational Quantum Regressor(VQR) is a hybrid quantum- classical algorithm for regression tasks. This involves predicting continuous numerical value from input data. The ZZ feature map is used to encode classical data into quantum state by utilizing both individual qubit rotations. In ZZ feature map **Entangling gates** are used.

ansatz

```
ansatz = real_amplitudes(num_qubits=num_features, reps=2)
ansatz.decompose().draw(output="mpl", style="clifford", fold=12)
```

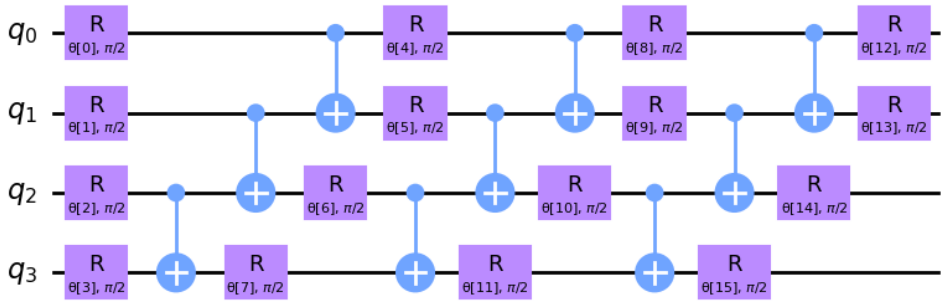


Figure 8. Configuration of Ansatz in VQR

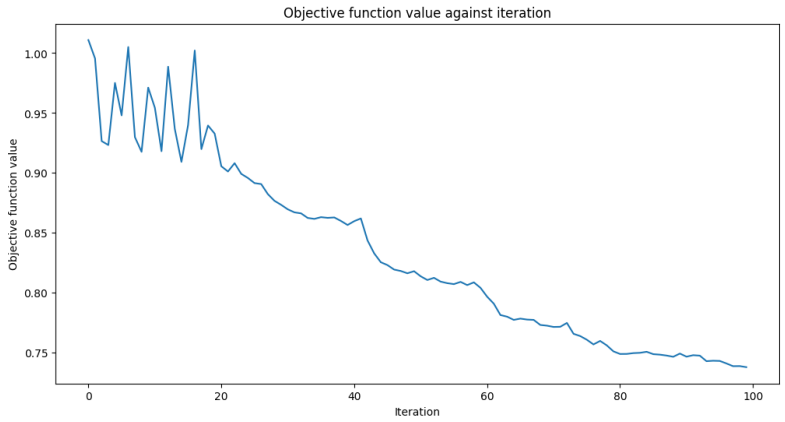
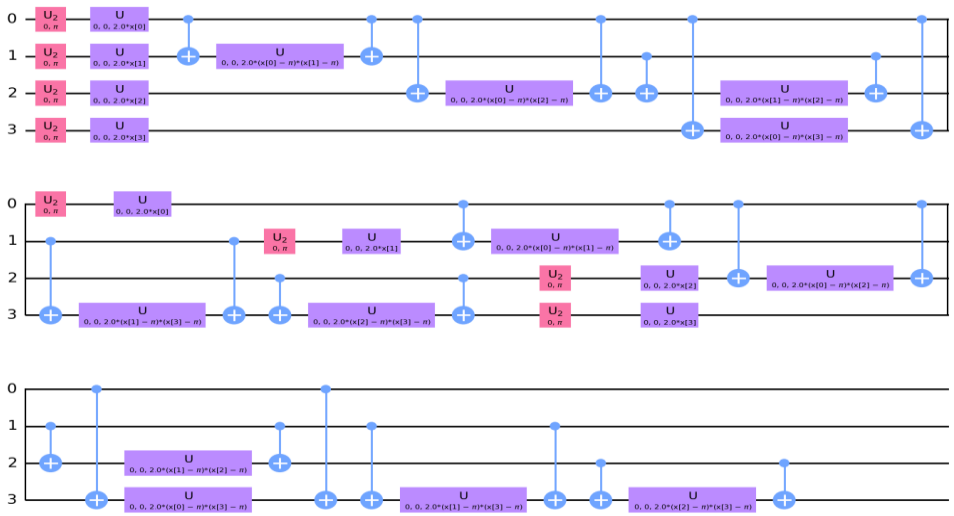


Figure 9. Iteration vs Objective function values

Regression Score: 0.26197676154461
Training time: 599 seconds
Final Optimized Angles: [1.0108438783101479, 0.9955128128634027, 0.9265578391716268, 0.9232838189192731, 0.9750862481997339, 0.9480439339605369, 1.0050242551149862, 0.9298663849723364, 0.9176311103320642, 0.9711838535107326, 0.9544238235323352, 0.9181541462010755, 0.988620876796387, 0.9367666185612759, 0.9092360831764599, 0.9400148511615837, 1.002121928779535, 0.9198824544964376, 0.9395554250197281, 0.9328088782050917, 0.9056034128072615, 0.9012148880505677, 0.9081515353205366, 0.8992338063619928, 0.8958176488908316, 0.8915356260346076, 0.8907356271853362, 0.8823656835299326, 0.8767574537232299, 0.873413362160888, 0.869665252234533, 0.8671419123004116, 0.8662914624968129, 0.8624690261559376, 0.8616892509032178, 0.8631358819490287, 0.8625640071284415, 0.8628878592878768, 0.8599619777558307, 0.856601196789177, 0.8597936000447443, 0.8620532288144117, 0.8437373416374837, 0.8329459330374269, 0.8255339322287144, 0.8231473403947647, 0.819414114450635, 0.818236282695455, 0.8163208375032804, 0.8180422576656488, 0.8138040388464134, 0.8107175402031156, 0.8125230955705789, 0.809366420522159, 0.8080666812982699, 0.8072782341012537, 0.8091363343863437, 0.8064783155647771, 0.8087448963970123, 0.8041711033402558, 0.7967840981356439, 0.7909777176321104, 0.7814800707374138, 0.7801401729163197, 0.7774593030452176, 0.7785049015164744, 0.7777273762116474, 0.7774465312058944, 0.7732204635600117, 0.7726225471420999, 0.7715604073319863, 0.7716966702385721, 0.7749157351162295, 0.7658304080856956, 0.7639585750728367, 0.7608894497399533, 0.7569834080560883, 0.7598993883083826, 0.7561928663086159, 0.7511949856266518, 0.7490113892365505, 0.7490578710398561, 0.7497276320012711,0.749989910927635, 0.75084330831974, 0.7488551591673921, 0.7484502476757807, 0.7477033570933564, 0.7466752835873202, 0.7493642238308545, 0.7467560828235233, 0.7480025947216054, 0.747625860989486, 0.7430190074112147, 0.7433733978660797, 0.7432099515214551, 0.7411184994320392, 0.73883487359892, 0.738908444280941, 0.7380232384553831]

Fig 10: Configuration of ZZ Feature Map used in VQR



5. Conclusion

5.1. Comparative Discussion: VQC vs. VQR Training Time

The observed difference in training time between the Variational Quantum Classifier (VQC) and the Variational Quantum Regressor (VQR) can be attributed to several algorithmic and computational factors. For the present DFIG analysis, the VQC model required **930 seconds**, whereas the VQR model converged in **599 seconds** for the same number of iterations.

First, **VQC involves probabilistic class label estimation**, which requires repeated circuit executions to accurately estimate measurement probabilities for each class. This increases the number of circuit evaluations per optimization step. In contrast, **VQR directly minimizes a continuous-valued cost function**, typically based on mean squared error, which is computationally less demanding in terms of measurement statistics.

Second, **VQC cost landscapes are generally more rugged and non-convex** due to the discrete nature of classification boundaries. As a result, the COBYLA optimizer requires additional function evaluations to approximate local linear constraints and ensure stable convergence. Conversely, the cost landscape in VQR tends to be smoother, enabling faster convergence with fewer circuit evaluations.

Third, the **number of effective measurements per iteration** is higher in VQC because class probabilities must be estimated with sufficient shot counts to reduce statistical fluctuations. VQR, on the other hand, is less sensitive to shot noise since regression outputs are averaged quantities, leading to reduced sampling overhead.

Finally, the **optimized parameter values** reflect distinct learning dynamics in the two models. VQC parameters converge toward larger angular values, indicating stronger entanglement and higher circuit expressibility, which increases circuit depth and execution time. In comparison, VQR parameters settle at relatively small angles, resulting in shallower effective circuits and reduced execution time.

Overall, these factors collectively explain the longer training time observed for VQC compared to VQR, despite using the same COBYLA optimizer and iteration count.

5.2 Performance score comparison with SVC and QVC: Variational Quantum Classifiers (QVC) underperform compared to Support Vector Classifiers (SVC). One of the reasons is Optimization Challenges (Training Difficulties) like Barren Plateaus, Slow Optimization and complex optimization surface.

Data encoding and circuit design like Z or ZZ feature maps, reps also play a role. Suitable modifications are likely to improve the performance.

In specific complex scenarios, QVC may outperform SVC . This is , after research in embedding and with high dimensional intricate , non-linear data patterns. **Quantum Support Vector Classifier** (**QSVC**) in Qiskit also likely improve the performance.

Declaration of Generative AI and AI-assisted Technologies

During the preparation of this manuscript, the authors used generative AI tools only to improve language and readability. The author take full responsibility for the content of the manuscript.

7. References

1. Ramakrishna Rao Mamidi, " Application of Neural networks for Prediction of Doubly Fed Induction Generator's Equivalent Circuit Parameters used in Wind Generators", *International Research Journal of Computer Science (IRJCS) Jan 2021*
2. Mamidi Ramakrishna Rao," Design Optimization of Doubly Fed Induction generator by Differential

Evolution", WASET, Dubai, January 30-31,2019.
3. Gonzalo-Abad. DFIG_Wind-Energy-Conversion-system
4. J.Tian, C.Su and Z.Chen, "Reactive Power Capability of the Wind Turbine with Doubly Fed Induction Generator". *Sino-Danish Centre for Education and Research, Denmark* *https://github.com/Theoodoh/DFIG_Wind-Energy-Converison-system-by-Gonzalo-Abad*
5. Mamidi Ramakrishna Rao, Jagdish Mamidi. "Multi-Objective Differential Evolution based Electromagnetic Design of Wind-Turbine Generator". *3rd World Congress on Wind and Renewable Energy ,(Wind Energy 2019) June 24-25,2019 Barrcelona , Spain.*
6. James Mc, Calley. "Double-fed electric machines-steady state analysis ". *A ppt presentation, IOWA State University*
7. "Training a Quantum Model on a Real Datase *02a_training_a_quantum_model_on_a_real_dataset.html* *https://qiskit-community.github.io/qiskit-machine-learning/tutorials/*
8. Yuxuan Du1,etc," Quantum Machine Learning", A Hands-on Tutorial for Machine Learning Practitioners and Researchers. *arXiv:2502.01146v1 [quant-ph] 3 Feb 2025*
9. Davis Arthur, "A Hybrid Quantum-Classical Neural Network Architecture for Binary Classication",*University of Florida, Florida, USA.,* Prasanna Date, *Oak Ridge National Laboratory, Tennessee, USA*
10. Data encoding: https://qiskit.qotlabs.org/learning/courses/quantum-machine-learning/data- encoding