Quantum Machine Learning, Variational Quantum Classifiers & Parameterized Quantum Circuits For Digital Sensor Signal Classification

Ryan Ueda Teo Shao Ming   
*Singapore Polytechnic*  
*School of Computing*Singapore  
ryanueda.21@ichat.sp.edu.sg

*Abstract* — The realm of Quantum Machine Learning (QML) has witnessed remarkable progress in recent years, akin to the rapid evolution of deep learning in the domain of artificial image generation. Originating in 2014, QML has emerged as a groundbreaking paradigm, offering novel avenues for solving complex classification problems. This paper embarks on a comprehensive journey through the landscape of QML, delving into essential aspects such as data preprocessing, model architecture experimentation, and the intricate mechanics underlying quantum classifiers. We elucidate the principles that power Quantum Machine Learning and unveil its burgeoning applications in the realm of classification, shedding light on its transformative potential in revolutionizing modern computational paradigms.

Keywords — *qis*ki*t, Quantum Circuits, Rotating Qubits, Variational Circuits, Quantum Neural Network Functions, Stochastic Gradient Descent, Loss Function, Learning Rate, Hyperparameter Tuning, Support-Vector Machine, SVM*

# **Introduction**

In the dynamic world of quantum computing and machine learning, the Variational Quantum Classifier (VQC) stands as a pivotal innovation that has reshaped the landscape of Quantum Machine Learning (QML). This introduction sets the stage by tracing the origins of Quantum Machine Learning, from its initial implementation to the transformative advancements brought forth by the VQC classifier.

Quantum Machine Learning marries the computational prowess of quantum computers with classical machine learning techniques, offering a quantum-powered solution to complex problems. The journey of QML began with Peter Shor's groundbreaking quantum algorithms, which unveiled the potential of quantum computers. However, it was the VQC that truly galvanized the field by introducing a novel approach to quantum-assisted classification tasks.

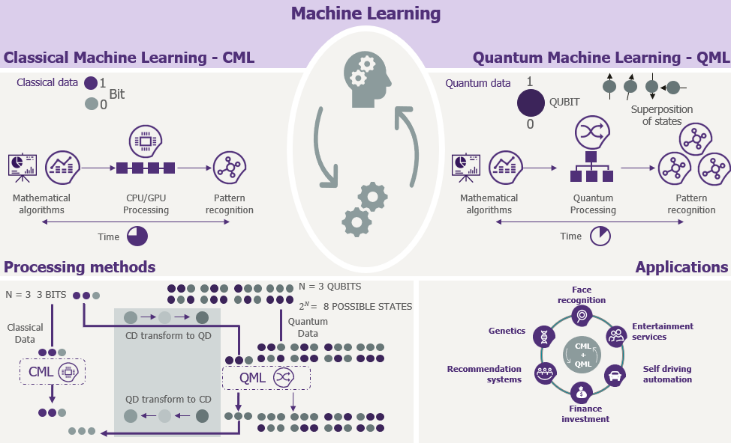
The VQC's brilliance lies in its fusion of quantum circuits and classical optimization. By parameterizing the quantum circuit and optimizing these parameters iteratively, the VQC efficiently solves a broad spectrum of problems, spanning from drug discovery to image recognition. In this paper, we will delve deeper into quantum machine learning for classification, specifically on digital sensor signals.

# **Quantum Machine Learning**

Quantum Machine Learning (QML) is a cutting-edge interdisciplinary field that explores the synergy between quantum computing and machine learning. At its core, QML aims to leverage the unique properties of quantum computers to enhance the capabilities of machine learning algorithms and solve complex problems more efficiently than classical computers.

There are several key aspects of QML: Qubits, Data Encoding, Variational Quantum Circuits & Quantum Algorithms.

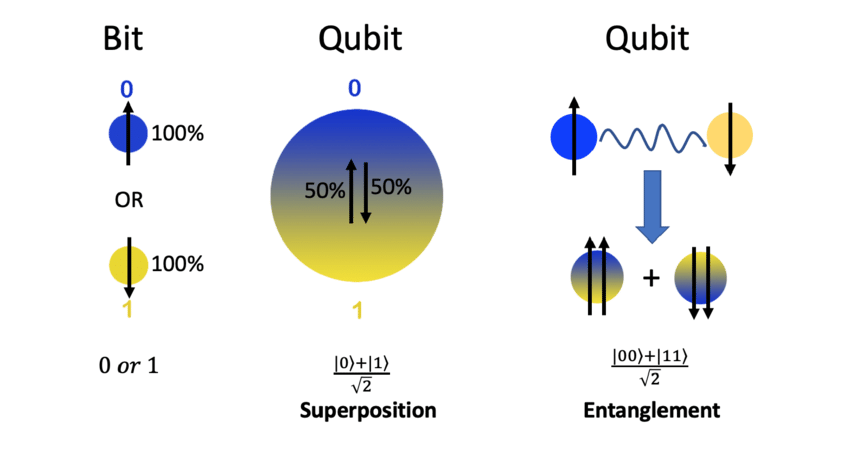
**Fig. 1. Classical Machine Learning VS Quantum Machine Learning**



We are all familiar with classical bits. These classical bits exist as either 0 or 1. A quantum bit, or qubit, is a superposition of 0 and 1. A single qubit therefore takes 2 classical values at once. Every operation on the qubit is done on both values at once. By this logic, you can consider every qubit to be 2 bits. This attribute of superposition allows for data to be processed much faster.

Additionally, qubits have the ability to be quantumly entangled, meaning a state where two or more qubits become correlated in a way where the state of one cannot be described independently of the other. Such attributes allow for quantum teleportation, allowing the state of one qubit to be transmitted to another qubit despite being a great distance apart.

**Fig. 2. Bits VS Qubits**



There are some key main differences between Bits and Qubits.

**Fig. 3. Key Differences Between Bits & Qubits**

|  |  |
| --- | --- |
| **Bits** | **Qubits** |
| Either 0 or 1 | Can have multiple values simultaneously |
| Does not follow superposition | Follows superposition |
| Value or State of a bit can be precisely determined; hence they are deterministic | Value or State of a qubit cannot be precisely determined; hence they are probabilistic |
| Implemented through electronic and optical devices | Implemented through quantum systems, such as ions, atoms, and superconductors |
| Circuit behaviour mimics classic physics | Circuit behaviour follows quantum mechanics |
| Operations use digital logic gates | Operations use quantum logic gates |

As most data is presented in classical bits instead of qubits, it must be encoded as qubits. This is necessary to obtain a quantum model. This process is otherwise known as data encoding, embedding, or loading, and is also the role of a Feature Map.

**Fig. 4. Quantum Feature Map**

A diagram of a data processing process

Description automatically generated

A quantum feature map encodes classical data to the quantum state space using a quantum circuit. The goal is to map each classical feature vector to a quantum state in the quantum Hilbert space. A quantum feature map typically involves a series of quantum gates and operations applied to an initial state. These can include Pauli-X, Pauli-Y, Pauli-Z, Hadamard or controlled gates. In some cases, quantum feature maps are parameterized, where they have adjustable parameters that can be optimized during training.

There exists a few main Quantum Feature Maps: ZFeatureMap, ZZFeatureMap & PauliFeatureMap.

**ZFeatureMap**

ZFeatureMap is used for encoding classical data into quantum states for quantum machine learning tasks, such as quantum support-vector machines (QSVM) and quantum data classification. By applying quantum gates that correspond to the Pauli-Z operator to each feature, it is able to encode classical data.

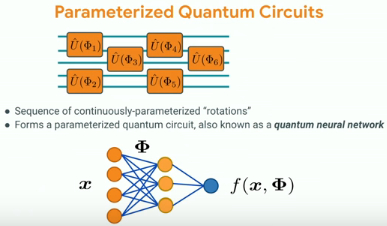
**ZZFeatureMap**

ZZFeatureMap is very similar to ZFeatureMap, with the main difference being that this feature map is designed to capture pairwise interactions between features.

**PauliFeatureMap**

PauliFeatureMap is designed to provide flexibility in encoding classical data into quantum states and is more versatile than the other two feature maps. A more fine-grained encoding of data into quantum states is made possible by applying quantum gates corresponding to various Paili operators to each feature or combinations of features in the data.

**Fig. 5. Parameterized Quantum Circuits**



Upon encoding our data in a quantum state, we must apply a parameterized quantum circuit. The circuit is similar to layers in a classic neural network, where it consists of a set of tunable parameters or weights. These weights are optimized to minimize an objective function. A parameterized quantum circuit is otherwise known as a parameterized trial state, variational form, or ansatz.

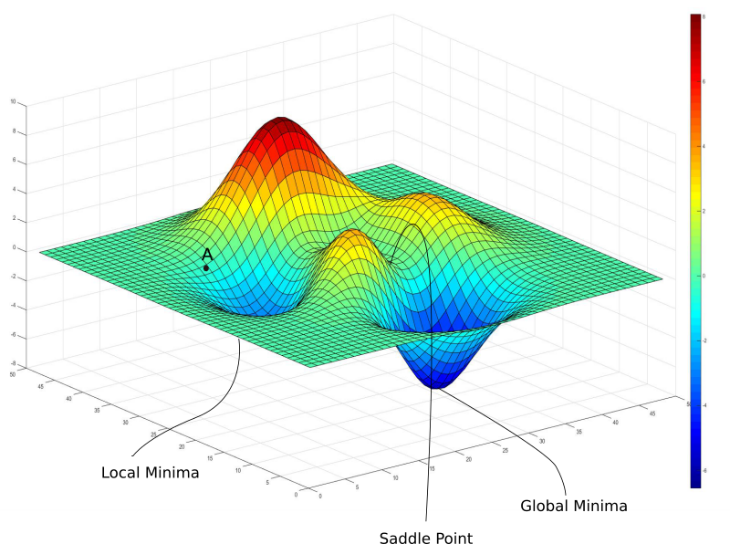
**Fig. 6. Optimizers For Quantum Networks**

A graph of numbers and numbers

Description automatically generated with medium confidence

In quantum networks, there are three commonly adopted optimization algorithms: Constrained Optimization By Linear Approximations (COBYLA), Simultaneous Perturbation Stochastic Approximation (SPSA), and Sequential Least SQuares Programming (SLSQP)

**Fig. 6. Classical Optimization Loop**



**COBYLA**

COBYLA is a derivative-free optimization algorithm, typically used for solving constrained optimization problems where the objective function and constraints may not be differentiable. By iteratively refining linear approximations, it is able to converge to an optimal solution. By approximating the objective function and constraints with linear models, it is well suited for non-linear, non-smooth and non-convex problems with box constraints and general inequality constraints.

**SPSA**

SPSA is a stochastic gradient-based algorithm, often employed in situations where obtaining accurate gradients is highly challenging, or computationally expensive. By estimating the gradient of the objective function using stochastic perturbations, it is robust to noise.

**SLSQP**

SLSQP is a gradient-based algorithm, that is designed for solving constrained problems where both the objective function, as well as constraints are differentiable. By iteratively minimizing a quadratic approximation of the objective function subject to linear and nonlinear inequality constraints, it is able to find a optimal solution while respecting the constraints.

**Fig. 7. Gradient Descent**

A graph of a function

Description automatically generated

While stochastic gradient descents are able to help us obtain the global minima for the loss function, it is not always the best strategy. For example, when the loss landscape has been squashed in the x-dimension, the initial point and learning rate is unable to find the minimum, due to the incorrect assumption that the loss landscape caries at the same rate with respect to the parameters.

**Fig. 7. Inaccuracies of Gradient Descent in a X-Dimensional Squeeze**

A diagram of a circle

Description automatically generated

**A couple of white rectangular objects with black text

Description automatically generated**

To solve this issue, we introduce Quantum Natural Gradients. The Quantum Fisher Information allows us to transform the steepest descent in the Euclidean parameter space to the steepest descent in the local model space. This is also known as the Quantum Natural Gradient.

**Fig. 8. Quantum Fisher Information & Quantum Natural Gradient**

A math equations and numbers

Description automatically generated with medium confidence

# **DATASET**

The dataset is composed of digital signals obtained from capacitive sensor electrodes that are immersed in water or in oil. Each signal, stored in one row, is composed of 10 consecutive intensity values and a label in the last column. The label is +1 for a water-immersed sensor electrode and -1 for an oil-immersed sensor electrode. This dataset should be used to train a classifier to infer the type of material in which an electrode is immersed in (water or oil), given a sample signal composed of 10 consecutive values.

he experimental setup is composed of a capacitive sensor array that holds a set of sensing cells (electrodes) distributed vertically along the sensor body (PCB). The electrodes are excited sequentially and the voltage (digital) of each electrode is measured and recorded. The voltages of each electrode are converted to intensity values by the following equation:

Where the Base Voltage is the voltage of the electrode recorded while the electrode is immersed in air. The intensity values are stored in the dataset instead of the raw voltage values.

The aim of the experiments is to get fixed-size intensity signals from one electrode (target electrode) when being immersed in water and oil; labeled as +1 (water) or -1 (oil). For this purpose, the following procedure was applied:

- The linear actuator was programmed to move the sensor up and down at a constant speed (20 mm / second).

- The actuator stops when reaching the upper and bottom positions for a fixed duration of time (60 seconds).

- At the upper position, the target electrode is immersed in oil; intensity signals are labeled -1 and sent to the PC.

- At the bottom position, the target electrode is immersed in water; intensity signals are labeled +1 and sent to the PC.

- The sampling rate is 100 msec; since each intensity signal contains 10 values, it takes 1 second to record one intensity signal

**Fig. 9. Sample Of Dataset**

A screenshot of a table

Description automatically generated

Some samples of the Low-Resolution and their High-Resolution counterparts shows the vast contrast in image quality between the two variants.

# **Exploratory Data Analysis**

Before we do any modelling, we must first perform some exploratory data analysis (EDA) to gain a better understanding of the data we are working with, as well as in order to gain further insight. To do this, we will first check for any NULL or missing values in the dataset.

**Fig. 10. df.isna.sum()**

A black background with white text

Description automatically generated

A quick check tells us that there are no missing values, making it a relatively clean dataset.

**Fig. 10. Pairplot Of Features**

A grid of graph paper

Description automatically generated with medium confidence

A simple pairplot of the features shows that across all columns, thee distribution and state of the data seems to be quite, similar, with low variance. This is to be expected, as all 10 features are sensor signals of the same nature.

# **Baseline**

In order to properly assess the quantum models, there has to be a baseline. In this project, we will be working with a classical machine learning model, something we all understand to a great degree. Hence, we will be using a vanilla Support-Vector Classifier.

The model is fitted with all default parameters, with nothing being done other than fitting the prepared data into the model. There is an observed accuracy of 96% for both train and test scores.

# **METHODOLOGY**

There are a few components that define the architecture of how we go about constructing the model from start to finish. As discussed earlier, we have to first convert our classical bit data to quantum qubits, before applying a quantum circuit. We then define an optimizer for the loss function and create a custom callback to track our model progress in real-time. Finally, we predict on the test data to evaluate the performance of our model.

**Fig 11. High-Level Visualization Of ZZFeatureMap**

A screenshot of a computer

Description automatically generated

After splitting our data into our train and test sets, we immediately apply a ZZFeatureMap to the data, to transform them into its quantum state. We can see that its constructed with multiple Hadamard and Pauli gates. If you look closely at the feature map diagram, you will notice parameters x[0], ..., x[3]. These are placeholders for our features.

**Fig 11. Low-Level Visualization Of ZZFeatureMap**

A lined paper with a blue object on it

Description automatically generated with medium confidence

To better understand the construction of our quantum circuit, we can call a function to decompose the gates into simpler forms. From here, we can see a single PauliGate.

**Fig 10. RealAmplitudes Parameterized Quantum Circuit**

**A diagram of a graph

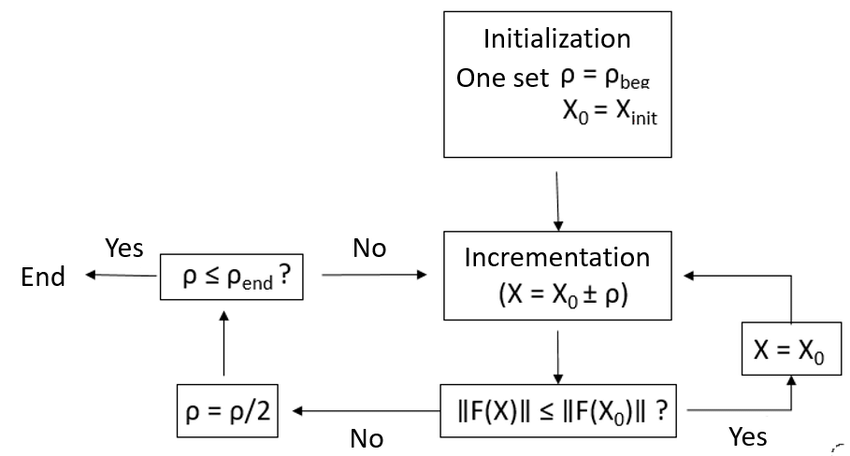
Description automatically generated**

From here, we apply a parameterized quantum circuit. This is otherwise known as a parameterized trial state, variational form, or *ansatz*. In this instance, we use the RealAmplitudes ansatz.

The RealAmplitudes circuit is a heuristic trial wave function used as Ansatz in chemistry applications or classification circuits in machine learning. The circuit consists of alternating layers of Y rotations and CX entanglements. The entanglement pattern can be user-defined or selected from a predefined set. It is called RealAmplitudes since the prepared quantum states will only have real amplitudes, the complex part is always 0.

There are a few benefits to using RealAmplitudes here. The most significant advantage is likely due to the way that it restricts parameters to be real numbers. This restriction reduces the parameter space compared to ansatzes with complex parameters. This leads to a simpler optimization landscape, making it easier to find optimal parameter values. In some cases, it can lead to faster convergence as well. Additionally, using real parameters reduces the amount of quantum resources required, such as qubits and gate operations. Lastly, its algorithmic simplicity provides a great starting point for quantum computing beginners.

**Fig 11. Constrained Optimization By Linear Approximations (COBYLA) Optimizer Scheme**



COBYLA is a derivative-free optimization algorithm. This means it does not require access to gradient information, which is useful as quantum circuit evaluation, where the output state and cost function are, measured, may not readily provide gradient information. On top of this, COBYLA treats quantum circuit evaluation as a black box, meaning it is not necessary for the algorithm to know the inner working of the quantum circuit. This allows for vast compatibility due to its lack of dependency on custom gradient computations. Below is a brief overview of how COBYLA works.

**Initialization**: COBYLA starts by initializing an initial guess for the optimization variables, which is typically denoted as "x."

**Approximate Linearization**: COBYLA maintains an approximation of the objective function in the vicinity of the current guess "x." It does this by forming a linear approximation of the objective function based on function evaluations at the current point and other points nearby.

**Move and Evaluate**: COBYLA selects a new point "x'" by perturbing the current guess "x" based on the linear approximation. This new point is chosen to minimize the estimated linear approximation of the objective function.

**Check for Convergence**: COBYLA checks for convergence conditions. If certain stopping criteria are met (e.g., the change in the objective function is sufficiently small), the algorithm terminates, and "x" is returned as the solution. Otherwise, it continues to iterate.

**Constraints Handling**: COBYLA also handles constraints on the optimization problem. It keeps track of constraints and ensures that the new points "x'" generated during each iteration satisfy the constraints. If the constraints are violated, COBYLA adjusts the step size to bring the solution back within the constraint boundaries.

**Iteration**: COBYLA continues the above steps iteratively until one of the stopping criteria is met. During each iteration, it builds and maintains an approximation of the objective function and tries to improve the current solution.

**Fig 12. Callback Functiuon For Objective Function**

**A screen shot of a computer code

Description automatically generated**

Now that we have defined the model, we create a custom callback function to help us visualize the objective function, allowing us to evaluate the performance of the model training in real time.

**Fig 13. Dimensionality Reduction: Principal Component Analysis (PCA)**

**A graph with blue and orange dots

Description automatically generated**

In an attempt to prevent the curse of dimensionality, we will use Principal Component Analysis (PCA) to reduce the number of features. We will then train both the vanilla SVC and quantum models to observe if there exists any stark differences.

# **Model Evaluation**

Now that the model has completed training, we will evaluate how the model has performed after running for 100 epochs. We will do this by first taking a look at the objective function, before predicting on the test set.

**Fig 14. Quantum Model: Objective Function Value Against Iteration**

**A graph with blue lines

Description automatically generated**

From our callback graph, we can see that the model trainig is generally a volatile process, with drastic spikes and drops despite only training for 100 epochs. In general the loss function has decreased over time, which is what we are looking for. Predicting on the test set gives us a train score of 69% and a test score of 66%, implying some possible form of underfitting.

**Fig 15. Quantum Model (Reduced): Discriminator & Generator Loss**

A graph with a line

Description automatically generated

This time, the objective function fluctuates extremely at the start, but quickly flatlines at around 40 iterations. This model returns a training score of 66% and a test csore of 63%, which is worse than before.

**Fig 16. EfficientSU2 Quantum Circuit**

**A diagram of a graph

Description automatically generated with medium confidence**

Just for fun, we will attempt constructing a third quantum model using a different parameterized quantum circuit from RealAmplitudes, EfficientSU2, known to be hardware-efficient. The EfficientSU2 circuit consists of layers of single qubit operations spanned by SU(2) and CX entanglements. This is a heuristic pattern that can be used to prepare trial wave functions for variational quantum algorithms or classification circuit for machine learning.

SU(2) stands for special unitary group of degree 2, its elements are 2×2 unitary matrices with determinant 1, such as the Pauli rotation gates.

On 3 qubits and using the Pauli Y and Z su2\_gates as single qubit gates, the hardware efficient SU(2) circuit is represented above.

**Fig 17. Quantum Model: EfficeintSU2**

A graph with blue lines

Description automatically generated

We can see here that the training is less volatile than the previous models, and drops in value much faster. It returns a train score of 66% and a test score of 64%. Although this is still inferior to the first quantum model, it is worth noting that it trains much faster.

# **CONCLUSION**

It's no surprise that classical machine learning models outperform their quantum counterparts at the moment. Classical ML has undergone significant development and refinement, while quantum ML is still in its early stages of maturity. Our results clearly indicate that the classical support vector machine (SVM) delivers the best performance.

When we reduced the number of features, as anticipated, all models' performance declined. Therefore, if computational resources allow, it's advisable to train your model on the complete feature-rich dataset without any feature reduction. However, we understand that resource constraints are common, and in such cases, you may need to strike a balance between dataset size, training time, and model performance.

Another noteworthy observation is that even a simple change in the ansatz architecture can lead to improvements in results. For instance, the two-feature model utilizing the EfficientSU2 ansatz outperformed the one using RealAmplitudes. This highlights the significance of hyperparameter selection in quantum ML, much like in classical ML. Therefore, it's worthwhile to invest time in searching for optimal hyperparameters, employing techniques like random/grid search or more advanced approaches commonly used in classical ML.

# **REFERENCES**

<https://ieee-dataport.org/open-access/dataset-binary-classification-digital-sensor-signals>

<https://qiskit.org/ecosystem/machine-learning/tutorials/02a_training_a_quantum_model_on_a_real_dataset.html#3.-Training-a-Quantum-Machine-Learning-Model>

<https://learn.qiskit.org/course/machine-learning/training-quantum-circuits>

<https://qiskit.org/documentation/stubs/qiskit.circuit.library.EfficientSU2.html>