Center Investigating the Effects of the QN-SPSA Optimizer on a VQC with an Efficient SU(2) on Convergence and Testing Accuracy

Introduction

Quantum computing has the potential to impact a broad array of industries, from finance to chemistry, to machine learning [0-2]. Theoretical computing speedups exist due to a quantum computer's ability to utilize the infinite-dimensional Hilbert space−which allows for generalizations of linear algebra and calculus[1]−offering 2^n simultaneously nonreal accessible states for every n qubits as opposed to 1 simultaneously real accessible state on a classical system. Not only do quantum computers scale exponentially in informational state capability, but they can also realize varying magnitudes, as long as they are orthogonal and normalized [5] as opposed to classical bits having one element of the wavefunction have a magnitude of ≈ 1 [0]. However, managing to produce a quantum state in practicality is extremely difficult due to the sensitivity of quantum states, the isolation of the quantum system, and the precise operations needed to control our qubits [2]. As a result, current quantum devices are riddled with noise and errors far above the threshold for fault tolerance [0-4]. Current quantum devices are also far too small to gain computational advantage outside of very specific instances. In this work, we will be using superconducting transmon qubits, a leading candidate in advancing quantum computers, and quantum simulators. Transmon qubits are quantum electrodynamic devices (QED), which implement electromagnetic fields in superconducting circuits [3]. What separates a transmon qubit from a charge-coupled qubit is the presence of a Josephson junction, a small insulator between Cooper pairs; this enables coherent quantum states to tunnel across the junction without extra voltage [3]. Transmon qubits are cooled at $< 10mK$; conduction electrons alternate between regions of high and low charge density, creating waves called phonons [3,5]; this enables infinite conductivity, or superconductivity [5]. Microwaves are sent through a gate capacitor, exciting the transmon qubit to an excited state.

There have been numerous models proposed to create a variational quantum classifier (VQC) that provides performance boosts over fair classical classifier algorithms [4, 1, 6]; yet, current VQC algorithms do not present quantum advantage and behave similarly to linear neural networks. In this work, we will focus on different optimizers as they apply to the performance of a variational quantum classifier; particularly the quantum natural simultaneous perturbation stochastic algorithm (QN-SPSA) because of its potential for scalability over other optimizers [7]. The efficient special unitary $SU(2)$ provides a simple, hardware-efficient ansatz (HEA) for our experiment [4] and the second-order Z evolution circuit applied to the adhoc dataset provides for effective implementation of higher-order data encoding [1]. The purpose of this research is not to demonstrate a quantum advantage over classical classifiers or to prove the supremacy of a particular optimizer but to explore how many steps it takes for the QN-SPSA to converge for small-scale applications on a basic VQC compared to similar optimizers.

Literature Review

For our experiment, it is important to review the theoretical fundamentals of the VQC, the QN-SPSA, and the simultaneous perturbation stochastic algorithm (SPSA); this is because we must know how to implement the theory on our quantum systems. Also, understanding the process behind the QN-SPSA, VQC, and SPSA allows for a more nuanced view of our findings.

Overview of Variational Quantum Classifier Circuit: Supervised Learning With Quantum-enhanced Feature Spaces

In [6], it is stated that a VQC will not have a quantum advantage if the feature vector kernel can be efficiently computed with a classical system. There are 4 main components of the VQC circuit: the encoding (feature map) circuit, the ansatz, the measurement scheme, and repetition [6]. First, we must take data from our dataset and project it into a quantum state:

$$|\psi(\overrightarrow{x_i})\rangle = \mathcal{U}_{\Phi(x_i)}|0\rangle.$$

x_i is every feature in the dataset, represented as elements in vector x, $\mathcal{U}_{\phi(x_i)}$ is the feature map, and $|0\rangle$ is the ground state of our qubits. The purpose of implementing $\mathcal{U}_{\phi(x_i)}$ is to project x_i into a fully separable feature space. This separation is significant as it dictates how the model will learn; failure to separate the data in the Hilbert space makes convergence difficult. Havlicek e al. used an Adhoc dataset for its convenient separability with the ZZ feature map [6].

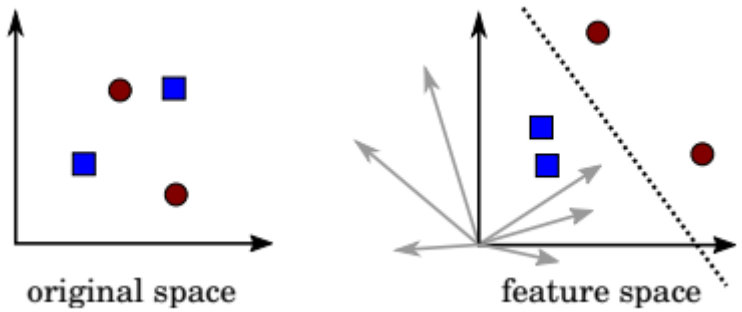


fig 1 Graphic representation of a feature map projecting data into a higher-order space [6].

n is our number of qubits, ϕ_s represents the coefficients of S, and S is the test set [6]. A basic support vector machine (SVM) is acting as our $U_{\phi(x_i)}$, and takes the following form [6]

$$U_{\Phi}(\overrightarrow{x}) = \exp(i \sum_{S \subseteq [n]} \phi_s(\overrightarrow{x}) \prod_{i \in S} Z_i).$$

However, $U_{\Phi}(\overrightarrow{x})$ on its own as a feature map is not sufficient, as it is easily classically simulable [6]. As a result, we can implement a feature space that takes advantage of quantum superposition [6] by applying Hadamard gates on $U_{\Phi}(\overrightarrow{x})$ such that

$$\mathcal{U}_{\Phi}(\overrightarrow{x}) = U_{\Phi}(\overrightarrow{x}) H^{\otimes n} U_{\Phi}(\overrightarrow{x}) H^{\otimes n}.$$

H is the Hadamard gate, and takes the form $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ [5]. In the instance of our circuit, ϕ_s has two elements. Our feature dimensions are n, and we use the following map for ϕ_s: $\phi_i(\overrightarrow{x}) = x_i$ After every entangling gate CX: $\phi_{\{1,2\}}(\overrightarrow{x}) = (\pi - x_1)(\pi - x_2)$. There are several ways to implement the feature map on quantum circuits, but Havlicek et al. used the aforementioned methodology as it is relatively shallow. This circuit shallowness is important as a higher circuit depth could lead to decoherence, causing the VQC to fail. Next, Havlicek et al. implemented a basic two-local as the ansatz due to its lightweight architecture [6]. The ansatz is the variational quantum circuit whose parameters will be optimized. As a rough analogy, every θ in the ansatz correlates to an artificial neuron in layer $\{i\}$. For the ansatz, Havlicek et al. used a two-local variational circuit [6] consisting of an $R_y(\theta_i)$ gate for each qubit followed by an $R_z(\theta_i)$ gate for each qubit [8] The $R_y(\theta_i)$ and $R_z(\theta_i)$ gates:

$$R_y(\theta_i) = \begin{pmatrix} \cos(\theta_i/2) & \sin(\theta_i/2) \\ \sin(\theta_i/2) & \cos(\theta_i/2) \end{pmatrix}, \; R_z(\theta_i) = \begin{pmatrix} \exp(-i\frac{\theta_i}{2}) & 0 \\ 0 & \exp(i\frac{\theta_i}{2}) \end{pmatrix}.$$

$R_y(\theta_i), R_z(\theta_i)$ are both matrix exponential of Pauli gates Y and Z, which are defined

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

So, $R_y(\theta_i) = \exp(-i\frac{\theta_i}{2}Y)$ and $R_z(\theta_i) = \exp(-i\frac{\theta_i}{2}Z)$. Afterward, a $CNOT$ gate is used for entanglement, which takes the form:

$$CNOT = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \end{pmatrix}.$$

After 2 repetitions of the aforementioned circuit, we now have $W(\overrightarrow{\theta})$, for $\overrightarrow{\theta}$ will be optimized by our optimization algorithms [6]. W is the unitary that is applied to the quantum state [6]. Our VQC supports two labels, mapped as $y \in \{0, 1\}$ [6]. The third step of VQC implementation is measurement [6]. We measure in the Z basis because we are using a binary readout [6]; the binary measurement is applied to $W(\overrightarrow{\theta})\mathcal{U}_{\Phi}(\overrightarrow{x})|0\rangle^n$. The expectation value is found over finding the empirical distribution of $\hat{p_y}(\overrightarrow{x})$, from $p_y = \langle \Phi(x)||W_\dagger(\theta)M_yW(\theta)||\Phi(x)i\rangle$ over many repetitions [6]. Now, the model's output can be fed into a classical optimization algorithm and the weights in $W(\overrightarrow{\theta})$ optimized. In [6], Havlicek et. al outlines 0, or label $[-1]$, and 1, or label $[+1]$ as outputs from the quantum system. In [6], Havlicek et al. used the SPSA with a cross-entropy loss function to optimize their VQC. After training and testing, Havlicek et. al found that the VQC converged and had a 95.45% accuracy on the testing set [6]. This review is significant as it outlines the general circuit architecture and signal flow of the VQC. We will be using this in our methodology as a guideline for prepping the dataset, ansatz, and feature map [6].

The SPSA Optimizer

In this work, we are using the QN-SPSA and the SPSA. The SPSA was chosen due to its performance and behavioral commonality with the QN-SPSA. Second, the SPSA was chosen due to the QN-SPSA simply being an SPSA that is approximated on a quantum natural gradient as opposed to a vanilla gradient [7].

The SPSA has garnered recent attention for its accurate approximation in working with high-dimensional spaces [9]. The SPSA is more suitable for applications such as annealing, as it is less precise than direct gradient approaches [9]. The SPSA is ideal in quantum computing as it allows for the input vector to be corrupted by noise, an enduring issue in noisy intermediate-scale quantum (NISQ) quantum devices [9]. Further, we can assure that the SPSA allows values of a noisy objective [9], a desirable quality for finding both the local and global minima. Minimization of a function's value while reducing the input parameters is an ongoing debate in computational sciences [7]. As demonstrated, Gacon et al. first defines the vanilla gradient that will be approximated. $f : \mathbb{R}^d \to \mathbb{R}$, let f be a cost function with d number of parameters. Where $\eta > 0$, η is the learning rate, let $\theta^{(0)}$ be the initial points, and $\nabla f(\theta^{(k)})$ with $\nabla f(\theta^{(k)}) \in \mathbb{R}^d$ denotes the vanilla gradient of f over k iterations [7]. The vanilla gradient is defined by the following:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla f(\theta^{(k)}).$$

However, the SPSA works a bit differently. First, it samples in a random direction $\Delta^{(k)} \approx \mathcal{U}(\{-1, 1\}^d)$, and then approximates the vanilla gradient [7], [9]. The gradient is scaled by some minute displacement, or perturbation, ϵ, and approximated by

$$\nabla f(\theta^{(k)}) \approx \frac{f(\theta^{(k)} + \epsilon\Delta^{(k)}) - f(\theta^{(k)} - \epsilon\Delta^{(k)})}{2\epsilon}\Delta^{(k)}.$$

A significant advantage presented is that the SPSA, as defined above, only needs 2 evaluations a step; this is much faster than the $\mathcal{O}(d)$ evaluations per step using a vanilla gradient. This method of optimization was used in Havlicek et al. with the VQC due to its noise resilience [6].

The QN-SPSA Optimizer

The QN-SPSA is similar to the SPSA, but instead of attempting to converge based on a vanilla gradient and consulting the Hessian each step, the QN-SPSA uses a 2-SPSA that approximates the quantum natural gradient (QNG) [7] and consults the quantum Fisher information matrix (QFIM) each step. While natural gradients require $\mathcal{O}(d^2)$ evaluations, QN-SPSA only requires $\mathcal{O}(1)$; this is a significant speed-up in exchange for accuracy [7], this is achieved by approximating the QFIM [7] of the ansatz and approximating the QNG. However, QN-SPSA models converge very sporadically; the cost per step reaches converges with much noise [7]. Demonstrated, the quantum natural gradient is similar, but the negative gradient $-\eta\nabla f(\theta^{(k)})$ is transformed to $-\eta g^{-1}(\theta^{(k)})\nabla f(p(\theta^{(k)}))$. $g^{-1}(\theta^{(k)})$ is the Reimann tensor from $p(\theta^{(k)})$, $p(\theta^{(k)})$ is the parameterized quantum model [7]. Unfortunately, computing g requires $\mathcal{O}(d^2)$ evaluations. To aid this issue, Gacon et al. demonstrate that we can replace $g^{-1}(\theta^{(k)})$ with a stochastic approximation $\bar{g}^k$, which only uses 4 evaluations. Unlike the SPSA, the QN-SPSA relies on the QFIM for each step [7]. The QFIM is consulted instead of the Hessian as a better approximation of the QNG. Gacon et al. state that we can calculate the QFIM by evaluating the Hessian of the

Fubini-Study metric tensor, which is practically equivalent to the QFIM up to a constant factor [7]. Ref. [7] uses a specific representation of QFIM to exploit the 2-SPSA (which uses two random directions as opposed to one in the SPSA), the Hessian of the Fubini-Study metric

$$g_{ij}(\theta) = -\frac{1}{2}\frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j}F(\theta,\theta')\Big|_{\theta'=\theta}.$$

i, j are the i-th and j-th elements of the parameter vector, θ. $F(\theta,\theta') = |\langle\psi(\theta)|\psi(\theta')\rangle|^2$, or the inner product of the conjugate transpose of $|\psi(\theta)\rangle$ and $|\psi(\theta')\rangle$, θ' is the conjugate of θ [7]. $F(\theta,\theta')$ is important as it will be used in the point-estimate

$$\hat{g}^k = -\frac{1}{2}\frac{\delta F}{2\epsilon^2}\frac{\Delta_1^{(k)}\Delta_2^{(k)T} + \Delta_2^{(k)}\Delta_1^{(k)T}}{2},$$

where

$$\delta F = F(\theta^{(k)}, \theta^{(k)} + \epsilon\Delta_1^{(k)} + \epsilon\Delta_2^{(k)}) - F(\theta^{(k)}, \theta^{(k)} + \epsilon\Delta_1^{(k)}) - F(\theta^{(k)}, \theta^{(k)} - \epsilon\Delta_1^{(k)} + \epsilon\Delta_2^{(k)}) + F(\theta^{(k)}, \theta^{(k)} - \epsilon\Delta_1^{(k)}).$$

It is important that the point-estimate $\hat{g}^k$ is smoothed into $\bar{g}^k$ to ensure that the estimate is positive [7] for purposes of optimization. To evaluate the Fubini-Study metric, we can evaluate the absolute value of the overlap between $|\psi(\theta)\rangle$ and $|\psi(\theta+\epsilon\Delta)\rangle$ [7]. This is the overlap between the ansatz and the ansatz with minutely shifted parameters. Gacon et al.'s work is relevant because it is a guideline for how to implement the QN-SPSA on a quantum machine learning model and why. We will be using Ref. [7] as an instruction for utilizing the QN-SPSA on our VQC.

How does the QN-SPSA optimize a VQC on a NISQ superconducting quantum computer? Ref. [7] uses a variational quantum Boltzmann machine (VarQBM) to benchmark the QN-SPSA as opposed to a VQC; the reason why the VarQBM was used is that Gacon et al. wanted to test the QN-SPSA on optimizing energy-based states [7]. The VarQBM is an energy- based quantum machine learning model while the VQC is not [7], [6]. In [6], the VQC is optimized using the SPSA. In this work, a new understanding is achieved as we are using a QN-SPSA to optimize the VQC as opposed to the SPSA. The importance of researching this knowledge gap is exploring how the QN-SPSA optimizes various domains of problems, the optimization problem in our case involves two labels: $[-1]$, $[+1]$.

Assumptions

1. We assume that the expectation value must be found over R repetitions because of non-deterministic quantum mechanics. According to the Copenhagen interpretation of quantum mechanics, there is a distinct 'quantum world' where quantum properties are only exhibited under certain conditions. Such is problematic as quantum mechanics is non- deterministic and quantum states collapse onto an unpredictable, classical state. This assumption is significant because it shows that $p_y(\vec{x})$ must be found over many repetitions.

2. The projection of the ZZ feature map and the chosen HEA will not exhibit unexpected behavior concerning the QN- SPSA and SPSA. This means that we assume that the VQC's change in performance between the two optimizers is a fault of optimization. This has significant effects on how we base our conclusions, we assume that if the model fails to converge or performs inadequately on the testing set it is a fault of optimization.

3. We assume that quantum mechanics is purely random. According to the Copenhagen interpretations of quantum mechanics, state collapse of a state in pure superposition over many trials will yield an approximately uniform empirical distribution. This is important as we assume that quantum processes are random, so we do not set a constant seed for any random properties of our quantum simulator.

Methodology

Isolating the effectiveness of an optimizer on a VQC is a rigorous experimental process. Successful implementation will require controlling all other aspects of the VQC but changing the optimization methods. After every step, results are stored in a log by callback. Each batch of shots to our quantum computer or

simulator branched into jobs. Every job is measured over 1024 shots; to which an expectation is calculated, 1024 repetitions are the default for quantum computing jobs. My access to quantum systems is limited, so it is important to reduce the number of shots to reduce valuable runtime, but not to the point of sacrificing the VQC's performance. The expectation values are fed into our varying optimizers, and a cost is found from a cross-entropy loss function that is used across both experiments.

Global Settings and Quantum Noise

While the adhoc dataset and initial optimization points are randomly generated, they have a controlled random seed of 3142; the specific choice of seed is arbitrary and will not affect the computation. Consequentially, all experiments will have the same dataset and initial points, allowing us to reproduce the behavior. Further, our quantum simulators will not have a constant random seed, as we want to mimic the random properties of our qubits in an attempt to have as realistic of a quantum simulator as possible. This means that `seed_simulator` and `seed_transpile` will not have fixed values. The device we submit jobs to will not change; switching devices from step to step will introduce confounding variables such as inconsistency in quantum architecture, wildly changing error rates, and qubit mapping conflicts. Due to device inhomogeneity, our ability to control quantum computers is limited due to precision in microwave control, readout, and coupling [2], [4]. As a result, our job counts are subject to noise and will hamper the model's performance when running on IBM quantum systems.

On the Use of Quantum Simulators

We will use QASM 3—a classical quantum simulator—for this experiment due to its accuracy of approximating IBM Quantum systems. Since IBM's quantum computers are a limited resource, using quantum simulators also provides a time- efficient solution. Second, quantum simulators allow us to set fidelity rates artificially; this enables us to explore how the VQC performs with low error rate systems and fault-tolerant quantum bits.

On the Use of *ibmq_belem*

ibmq_belem will be used in experimentation. *ibmq_belem* is a 5 qubit, superconducting transmon quantum computer housed in a dilution refrigerator at the IBM T. J. Watson Research Center in Yorktown Heights, NY [10]. *ibmq_belem* will be accessed through IBM Quantum with my IBMid. The reason we will use a real quantum system is that we want to see how the VQC will perform with current, noisy quantum devices. *ibmq_belem* specifically will be used due to my access and runtime restrictions; nonetheless, *ibmq_belem* will be sufficient for this experiment as it is a usable, universal quantum computer.

Adhoc Dataset Generation to Avoid Confounding Variables

The purpose of using an adhoc dataset is to isolate the performance of the optimizer. The adhoc is useful because it is a toy dataset designed to have the ability to be fully separated from the second-order Z evolution circuit that will be used to encode our data in the preprocessing stage of the model. This is so as we are not examining the model on the encoding of the ZZ feature map. So, we can eliminate the encoder's ability to accurately project the dataset into the feature space, affecting the performance of the model. In generating the dataset, we have a gap for clear separability, $\Delta = 0.3$, and the dataset and classification set will consist of 20 training examples, complete with data points and labels. The data has two dimensions for the two feature dimensions in the ZZ feature map and has one real number per dimension ranging in value from 0 to 7 for x_1 and 0 to 6 for x_0. We use 20 to not overtrain our model, and generate no more data points than necessary. There are 20 testing examples to benchmark the model post-training, it is standard for the testing set to be smaller than the training set. This testing phase is crucial because it shows how the model performs on the data it hasn't seen before. Occasionally, models begin to memorize the training dataset in a phenomenon known as overfitting [11]. In the instance of overfitting, the model performs better than the testing set; this means it has learned noise in the training set that is irrelevant to classification.

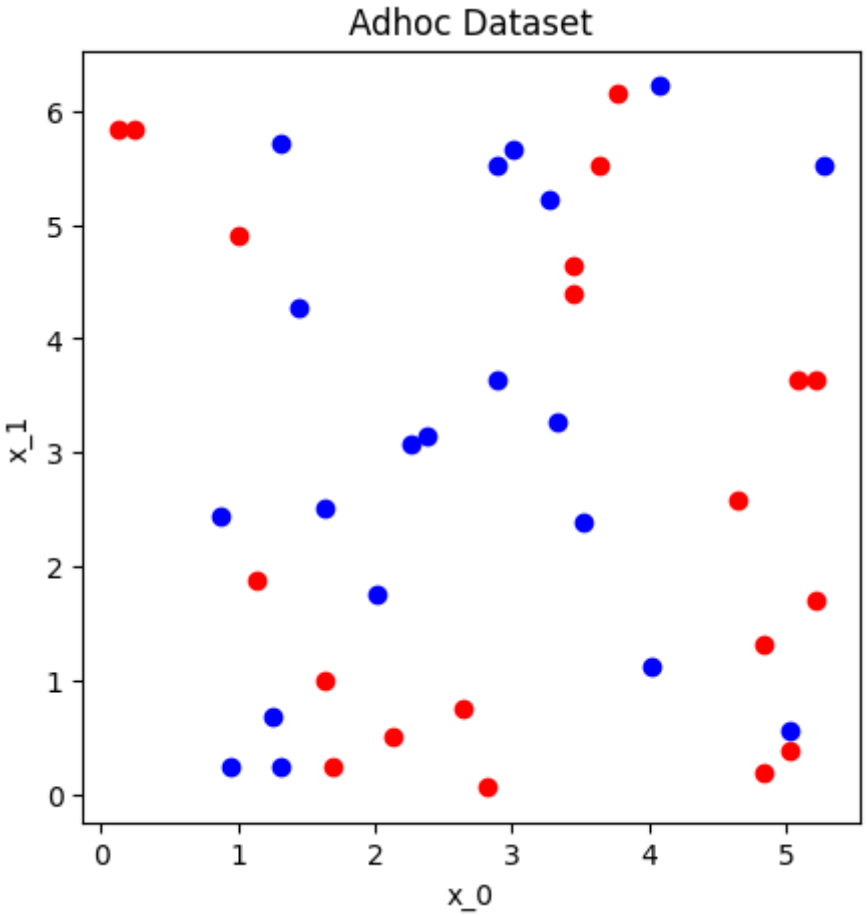


fig 2 Plot of adhoc dataset x_1 v. x_0, blue data point correlates to a label of 1, while red correlates to a label of 0.

Higher-Order Encoding with the Second-Order Z Evolution and Preprocessing

The ZZ feature map will act as a quantum 'cast' for our efficient $SU(2)$, mapping the classical adhoc dataset to a higher-order space. The two feature dimensions are mapped below as $x[0]$ and $x[1]$ respectively to fit the arguments of the ZZ feature map. Our adhoc dataset is encoded via SciPy's `OneHotEncoder` for compatibility with the ZZ feature map. `OneHotEncoder` will take every label and map it to a column, showing a `True` or `False` for the data in the form of ones and zeroes. Since the adhoc dataset already has labels of 0 and 1, the `OneHotEncoder` merely transposes the list of labels to be categorized vertically. The ZZ feature map is used because it is a gate implementation of $\mathcal{U}_\Phi(\vec{x})$, enabling clear separability as proven in the review of Ref. [6].

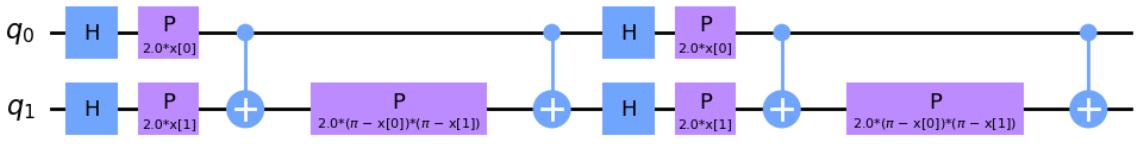


fig 3 Circuit demonstration of ZZ feature map. The circuit is repeated twice to have a greater span over the Bloch sphere.

Efficient $SU(2)$ Ansatz

Our $SU(2)$ repeats twice so we can encode more parameters into our ansatz, allowing for a more complex model, but not complex to the degree of vanishing gradients. The $SU(2)$ was chosen as it is a basic two-local circuit, minimizing the probability that a niche property of the ansatz architecture affects the convergence of the

model from the optimizers. The $SU(2)$ has a circuit width of 2 for the 2 feature dimensions in the ZZ feature map. This is the last circuit block in the variational model before measurement. This is analogous to the hidden neurons in a dense neural network.

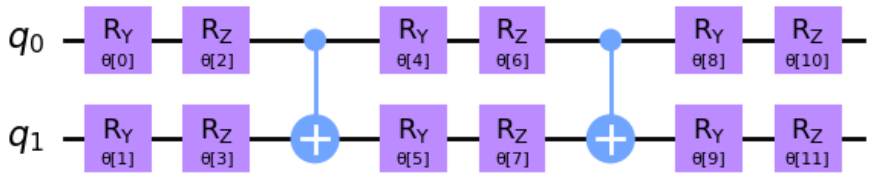


fig 4 Circuit representation of the ansatz.

Each $\theta[i]$ correlates to an element in θ_i, allowing for straightforward implementation of QN-SPSA and SPSA optimization as presented in Ref. [7].

Measurement Scheme

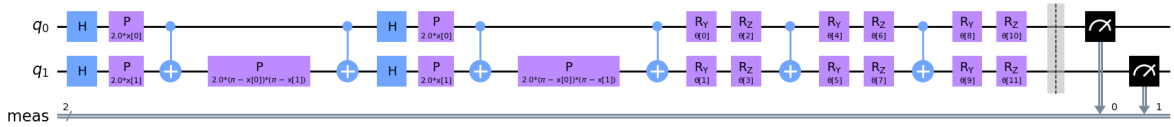


fig 6 Composed circuit of VQC with measurement.

We cannot classically train quantum states, so we must collapse the quantum state with measurements in the computational basis [7], [5]. However, a single measurement will not be sufficient to calculate the expectation value [7]. So, we repeat the process many times to find the empirical distribution. We find this empirical distribution so we can calculate the cost of the step. Once the cost has been calculated, it can be fed through the loss function f, as presented in Ref. [7].

Optimization and Evaluating the QFIM

The VQC's settings are changed only when switching between optimizers. Code is changed: `SPSA.minimize(**args)` vs `QNSPSA.minimize(**args)`. This is because we want to isolate the effect of the optimizers on the VQC. However, there is another step of the QN-SPSA because it is a 2^{nd} order SPSA that consults the QFIM as opposed to the Hessian of the ansatz [7]. Since the QN-SPSA consults the QFIM with every iteration, the QFIM must be evaluated. There is a practical method of approximating the QFIM experimentally. Take the circuit of the ansatz and evaluate the empirical distribution over 1024 shots. Then, nudge the parameters in the ansatz by $+\epsilon\Delta$ and evaluate the empirical distribution over 1024 shots. Now, find the overlap between the first and second evaluations; we now have the Fubini-Study metric as presented in Ref. [7]. In Qiskit, this is implemented as `fidelity=QNSPSA.get_fidelity(model.ansatz, QuantumInstance(backend), expectation=PauliExpectation())`. For both the QN-SPSA and SPSA, $\epsilon = .01$ and $\eta = .01$. These values are chosen as they are appropriate for this problem. Keeping the learning rate and perturbation from changing eliminates a potential confounding variable; assuming no special interactions between the data fed from the ZZ feature map and the ansatz. It is to be noted that the cost function, f, will be a cross-entropy loss function; this loss function was chosen as it efficiently aids in optimization for classification problems[11].

Testing

After training, the VQC is tested on the test set, which is preprocessed in the same method as the training set. Since we are not changing the weights in $W(\vec{\theta})$ as we are not training the model, the model is fed forward to each example without consulting the optimizer after each step. After all testing set examples are run through the

VQC, each predicted label, $\hat{y}$, is compared with the actual label, y. If $\hat{y}$ and y agree, the prediction is correct. Finally, we calculate the accuracy of the model by dividing where S_i agrees with y by the number of elements in S. The testing accuracy is compared across the two trials. We have a testing phase because it shows us how the model performs on data it has not seen before. Second, testing lets us know if the model is overfitted or underfitted [11].

Hypothesis

I hypothesize that the VQC optimized with the QN-SPSA will converge faster than the VQC optimized with the SPSA on all trials, but will have a slightly lower testing accuracy. I predict that the QN-SPSA will converge similarly to the VarQBM in Gacon et al. [1]. In both trials, the quantum system will be costlier and less accurate than its simulated counterpart. I also predict that the QN-SPSA will have the most varied path to convergence.

Data Analysis

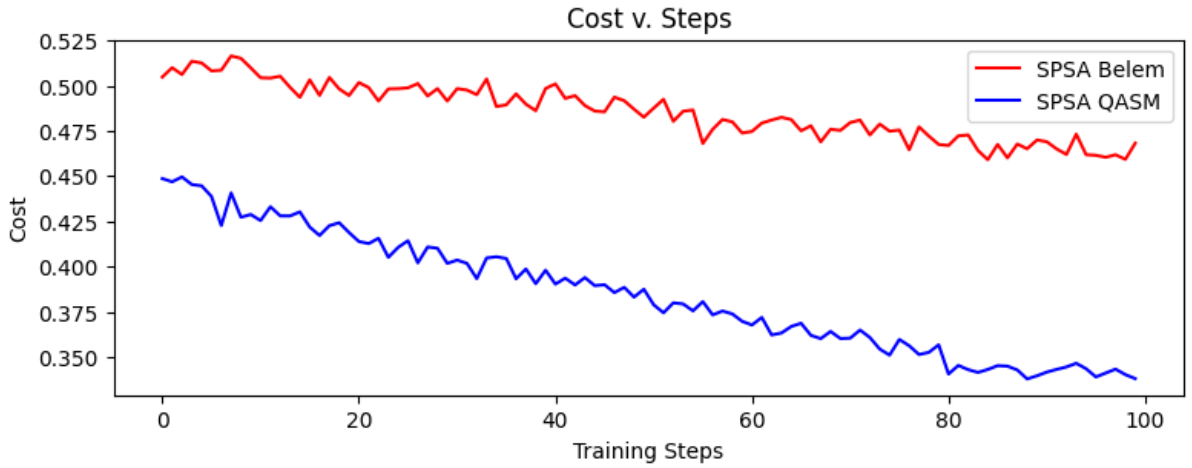


fig 6 VQC optimized by SPSA, cost v. training steps. SPSA Belem is the results of the real quantum system, while SPSA

QASM is the simulator results. The SPSA converged on the simulator at about 80 steps. The SPSA on the quantum system lowered in cost much more slowly than the simulated SPSA and had a higher initial cost. This is to be expected, the quantum system is much nosier than our chosen simulator. However, both models had a 90 % accuracy on the testing set. This defied my hypothesis that the quantum system would perform worse on the test set. The SPSA also demonstrated its noise-resilient properties in the testing set; despite having a more sporadic and unstable training process, it performed as well as the quantum simulator. As expected, the highest costs were in the initial iterations; the VQC is far from the global minima in the initial steps.

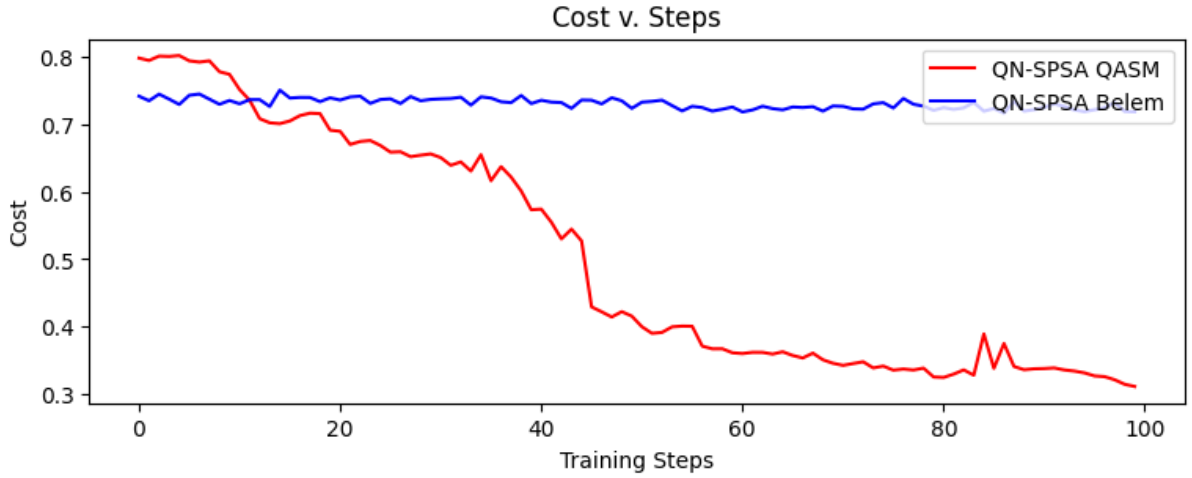


fig 6 VQC optimized by QN-SPSA, cost v. training steps. QN-SPSA Belem is the result of the real quantum system, while QN-SPSA QASM is the simulator result.

The results of the QN-SPSA trials also defied my hypothesis. The QN-SPSA on the simulator converged much faster than any of the experiments run. However, the VQC optimized with the QN-SPSA on the quantum simulator overtrained as it had a test accuracy of 85% despite having the lowest final cost. The VQC trained on the QN-SPSA completely failed to converge, and had the highest mean cost across all trials at .744. The test accuracy reflected the con-convergence, with a test accuracy of 45%, the lowest of any trial in this work. My hypothesis was defied by my second prediction regarding the QN- SPSA's behavior with the VQC: the QN-SPSA-trained VQC run on *ibmq_belem* had very little variance across the training period. This is most likely due to the failure to converge.

Conclusion

The results of this experiment defied my hypothesis. While the simulator run of the model trained with the QN-SPSA converged faster than any of the trials in my experiment, the QN-SPSA-trained model on IBM-Q Belem failed to converge. Oddly, the simulated QN-SPSA overtrained as evidenced by the lowest final costs and weakened test accuracy. On the real quantum system, the QN-SPSA-trained model's non-convergence was likely due to a few potential factors: noise resilience of the QN-SPSA, the Barren Plateau, and potentially inappropriate repetitions.

Despite our efforts to avoid the Barren Plateau, it is possible that we have run into it anyway on the VQC trained with a QN-SPSA. While the exact origin of the Barren Plateau is under scientific debate—it presents itself as a sudden, and seemingly random, vanishing gradient [1]. It is clear that the VQC trained with the QN-SPSA on the quantum system did not move; the cost did not change across all 100 iterations and the test set accuracy was no better than guessing, (note that the testing set had similar numbers of each label.) For further work, we could repeat the methodology in this work, but change the cost function to be local rather than global while keeping the depth of the circuit constant [12]. Thus, the ansatz is shallow [12]. Unfortunately, this 'shallow' approach makes the experiment easily simulated on classical systems, diminishing quantum advantage [12].

Spall's SPSA is known to be noise resilient [9]. It is possible that the QN-SPSA is not as noise resilient, given certain optimization problems. This could mean the QN-SPSA has a noise threshold where it begins to experience a vanishing gradient. We can come to this conclusion because the QN-SPSA only performed worse than the SPSA when the model was trained on a real quantum system. To support this, Gacon *et al.* performed the QN-SPSA on a VQA and a Boltzmann machine on *ibmq_montreal*, which has a μ_H(mean of H gate error) of .035% and a μ_{CNOT}(mean of $CNOT$ gate error) of 1.022 [13], while *ibmq_belem* has a μ_H of .039% and a μ_{CNOT} of 1.399%; this means that the system that Gacon *et al.* used for their experiment was less noisy. Gacon *et al.* found a significant convergence speedup with the QN-SPSA in those two models [7]. Further work can be done to determine the noise threshold of the QN-SPSA by using a similar methodology presented in this work without the real quantum system and SPSA trials. Instead of changing the order of SPSA, we can use quantum a simulator with varying degrees of simulated noise. Afterward, analyze the relationship between noise and iteration of convergence by determining which gate and measurement error rates the VQC fails to converge.

In reflection, certain aspects of this experiment could be done differently for a better understanding of QN-SPSA optimization with the VQC. For one, more data could be used at the expense of longer computation time. While I had 20 samples in the test set due to current limitations of my access to *ibmq_belem* and lack of time, more testing samples would yield a more accurate depiction of the VQC's testing accuracy. Second, using *ibmq_montreal* or a device with similar or better fidelity would have yielded a better-performing VQC. Another limitation is that we only used *ibmq_belem*, further work can be done by repeating our experiment, but changing the real quantum system, then analyzing the iteration of convergence and testing accuracy with each device. Further analysis could be done with the results of my experiment. For example, a kernel matrix could be analyzed by comparing the kernel trained with a quantum and the kernel trained with the quantum simulator. Then, compare the differences between kernels trained with the QN-SPSA vs. with the SPSA.

The implications of this work show the QN-SPSA's effectiveness of training a VQC on a superconducting NISQ quantum system with a $SU(2)$ ansatz and a ZZ feature map. But, further work needs to be done to truly

determine the effectiveness of the QN-SPSA on a VQC. Notably, the field of quantum machine learning has existed for no longer than a decade. So, every part of the quantum machine learning process—from SVMs to the optimizers—is in hot debate among the scientific community. The potential of quantum machine learning is strong, and the approach of which quantum machine learning methodology is superior remains uncertain.

Citations

[0] Carena, Marcela and Lamm, Henry and Li, Ying-Ying and Liu, Wanqiang, *Improved Hamiltonians for Quantum

Simulations of Gauge Theories*. vol.129, issue 5, 051601. American Physical Society. (2022)

[1] McClean, J.R., Boixo, S., Smelyanskiy, V.N. *, et al. Barren plateaus in quantum neural network training landscapes*. Nat

Commun **9**, 4812 (2018). https://doi.org/10.1038/s41467-018-07090-4

[2] Connell, Ghadimi, Blums, Norton, Fisher, Amini, Volin, Hayden, Pai, Kielpinski, Lobino, and Streed, *Scalable Ion-

Photon Quantum Interface based on Integrated Diffractive Mirrors* in Frontiers in Optics. FM3E.5. OSA

Technical Digest. (2017) [3] Roth, Ma, Chew, *An Introduction to the Transmon Qubit for Electromagnetic Engineers*. arXiv:2106.11352v1. arXiv

Preprint. (2007) https://doi.org/10.48550/arXiv.2106.11352

[4] Funcke, Hartung, Heinemann, Jansen, Kropf, Kühn, Meloni, Spataro, Tüysüz, Yap, *Studying quantum algorithms for

particle track reconstruction in the LUXE experiment*. arXiv:2202.06874. ArXiv Preprint. (2022)

https://doi.org/10.480/arXiv.2106.11352

[5] Wilde, *From Classical to Quantum Shannon Theory*. Hearne Institute for Theoretical Physics. (2021)

[6] Havlicek, Corcoles, Harrow, Temme, Kandala, Chow, Gambetta, *Supervised learning with quantum-enhanced feature

spaces*. vol. 557 pg. 209-212. Nature. (2019) https://doi.org/10.1038/s41586-019-0980-2

[7] Gacon, Zoufal, Carleo, Woerner, *Simultaneous Perturbation Stochastic Approximation of the Quantum Fisher

Information*. vol.5, pg. 567. Quantum. https://doi.org/10.22331/q-2021-10-20-567

[8] Nachtergaele, *Introduction to Quantum Spin Systems*. UC Davis.

https://www.math.ucdavis.edu/~bxn/introduction_to_qss-lecture4-su2.pdf

[9] Spall *An Overview of the Simultaneous Perturbation Method for Efficient Optimization*. vol.19, no.4. Johns Hopkins

APL. (1998) https://secwww.jhuapl.edu/techdigest/content/techdigest/pdf/V19-N04/19-04-Spall.pdf

[10] IBM Quantum Research. https://research.ibm.com/quantum-computing

[11] Brownlee, *A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks*. Machine Learning Mastery.

(2019) https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

[12] Cerezo, Sone, Volkoff, Cincio, Coles, *Cost Function Dependent Barren Plateaus in Shallow Parametrized Quantum

Circuits*. arXiv:2001.00550. ArXiv Preprint. (2020) https://doi.org/10.1038/s41467-021-21728-w

[13] McClure, Gambetta, *Hitting a Quantum Volume Chord: IBM Quantum adds six new systems with Quantum Volume 32*.

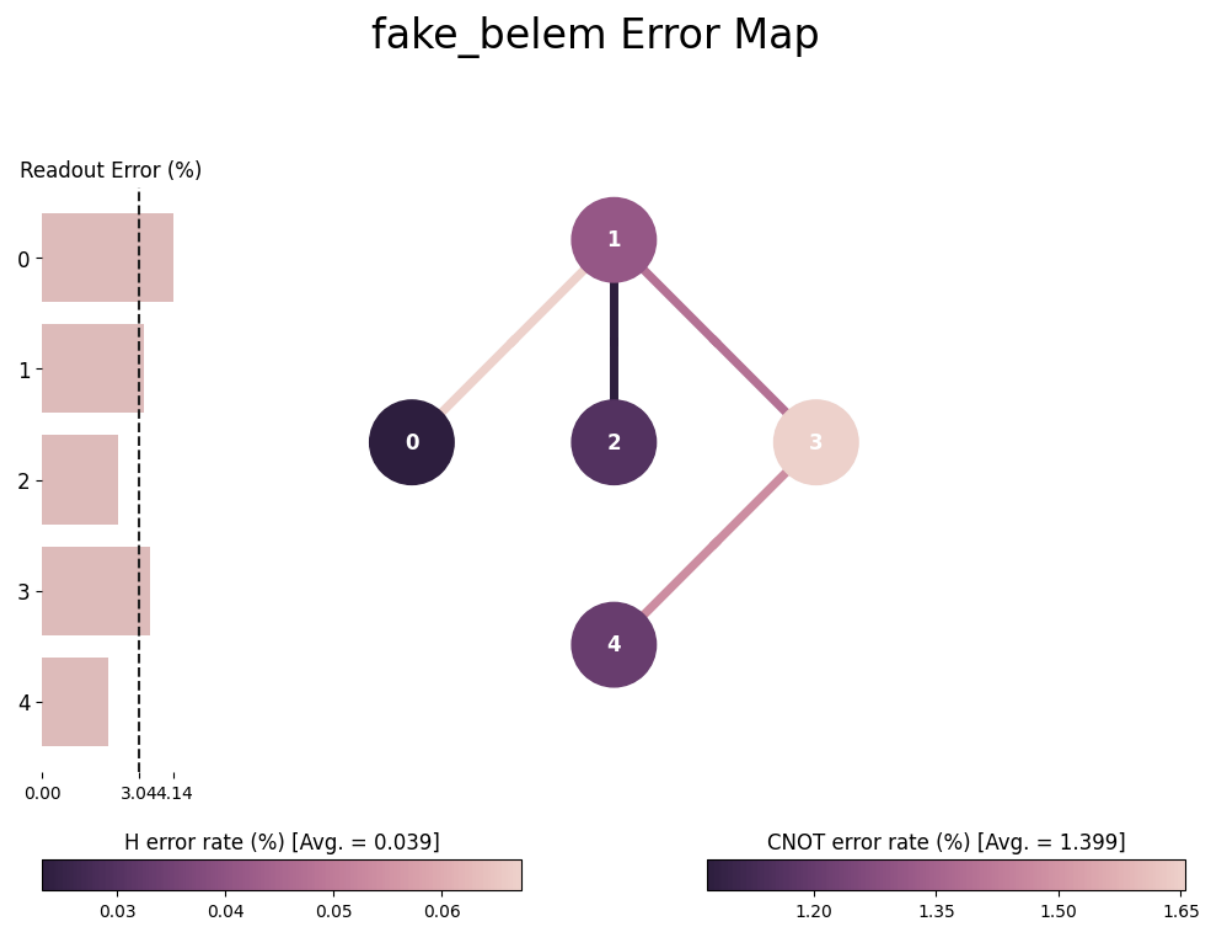
Quantum Computing. IBM Quantum. (2020) https://www.ibm.com/blogs/research/2020/07/qv32-performance/

Appendix

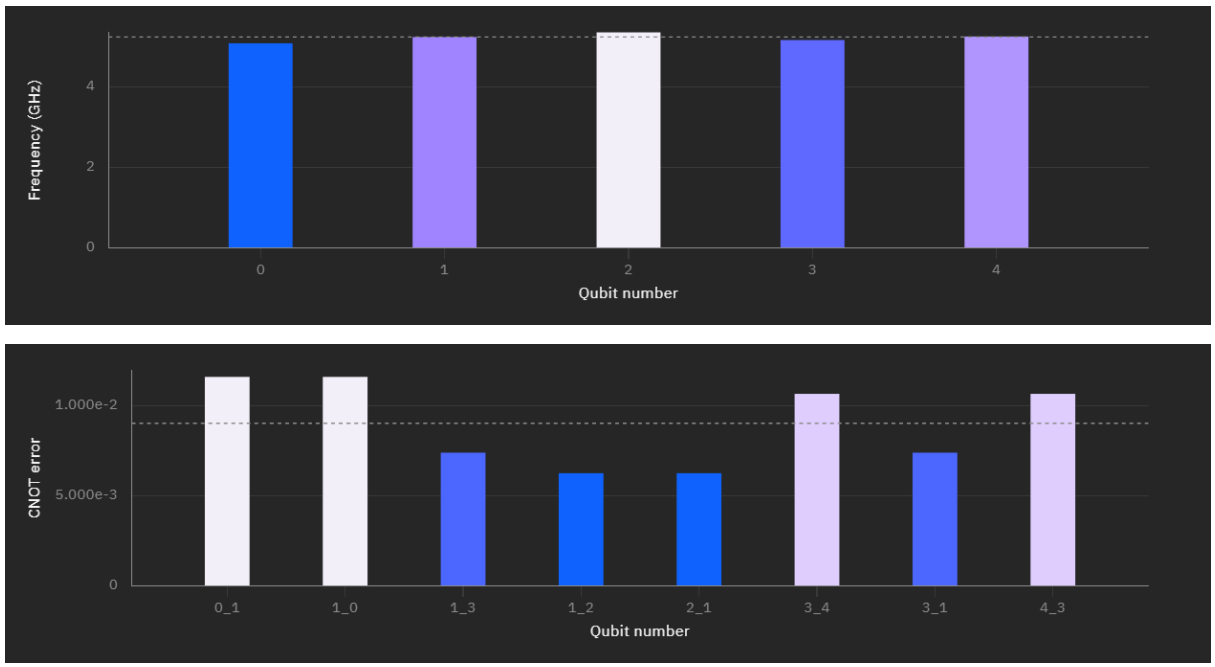
1. Raw Dataset

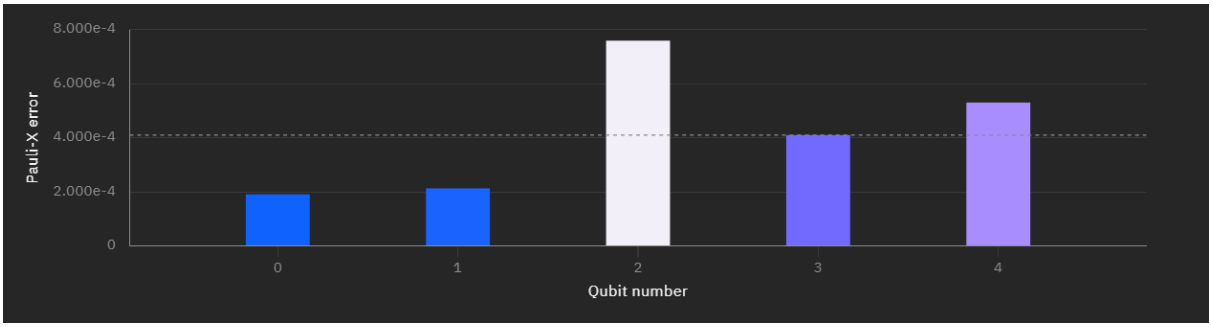
values: [0.9424778 5.0893801] : 0
values: [1.00530965 2.19911486] : 0
values: [5.59203492 3.45575192] : 0
values: [0.18849556 3.89557489] : 0
values: [2.38761042 4.58672527] : 0
values: [0.43982297 2.32477856] : 0
values: [4.20973416 3.20442451] : 0
values: [1.63362818 0.62831853] : 0
values: [4.0212386 2.136283] : 0
values: [2.32477856 0.   ] : 0
values: [1.82212374 0.12566371] : 0
values: [2.26194671 1.00530965] : 0
values: [2.07345115 1.38230077] : 0
values: [4.96371639 3.39292007] : 0
values: [5.78053048 1.38230077] : 0
values: [1.50796447 3.01592895] : 0
values: [4.08407045 5.65486678] : 0
values: [5.27787566 4.20973416] : 0
values: [4.33539786 1.38230077] : 0
values: [3.89557489 0.81681409] : 1
values: [3.0787608 4.33539786] : 1
values: [5.02654825 3.95840674] : 1
values: [2.136283   3.45575192] : 1
values: [3.45575192 3.26725636] : 1
values: [4.20973416 0.81681409] : 1
values: [0.75398224 3.33008821] : 1
values: [1.44513262 3.76991118] : 1
values: [0.9424778 3.51858377] : 1
values: [1.38230077 2.45044227] : 1
values: [3.01592895 3.83274304] : 1
values: [5.52920307 2.136283] : 1
values: [2.70176968 3.89557489] : 1
values: [1.13097336 5.59203492] : 1
values: [1.31946891 2.26194671] : 1
values: [5.71769863 1.75929189] : 1
values: [1.88495559 5.34070751] : 1
values: [3.39292007 1.19380521] : 1
values: [3.26725636 2.0106193] : 1
values: [4.20973416 3.83274304] : 1

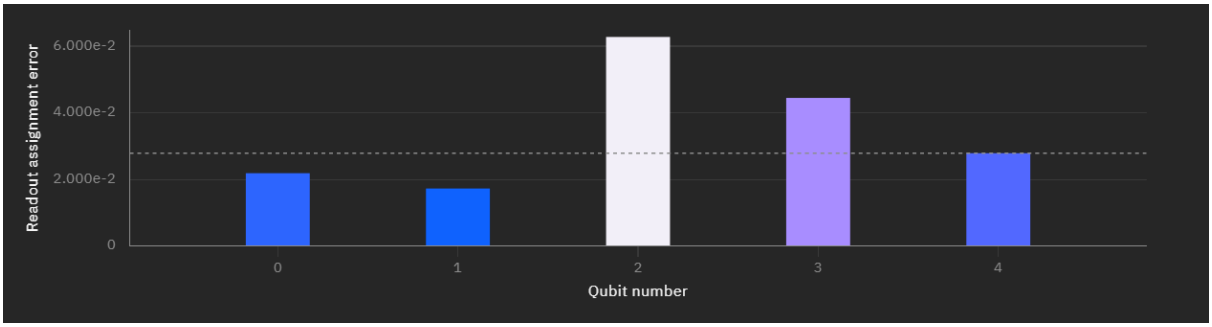
2. *ibmq_belem* Fake Configuration



3. *ibmq_belem* Calibration Data **




8.000e-4
6.000e-4
4.000e-4
2.000e-4
0
Pauli-X error
0
1
2
3
4
Qubit number


6.000e-2
4.000e-2
2.000e-2
0
Readout assignment error
0
1
2
3
4
Qubit number