CPEN400Q Final Report

Claire Song, Justin Hua, Matthew Chow, Matthew Yen

April 2023

Contents

1 Theory

1.1 Purpose

In today's world, classical computers contain hardware with the ability to perform error correction without requiring any software-side modifications. Error correction is the process where the hardware performs certain routines and techniques that automatically mitigate any noise errors on the physical media. Error correction is particularly crucial for quantum computers because these tend to have larger error rates compared to classical computers. This is due to a concept called decoherence which increases with the number of quantum computations performed. To reasonably remain within fault tolerance thresholds, which define the limits of reliability for quantum computers, researchers must find more efficient quantum correction algorithms.

Current error correction algorithms however, face major issues when it comes to the phenomenon of barren plateaus. Namely, the cost function gradient decays exponentially as the number of qubits in the system increases, until we are left with an essentially untrainable model. In the general case, it is also challenging to determine the encoding and noise correction unitary gates to apply, especially since these are heavily reliant on the type of noise that is present in the system. The application of Variational Quantum Algorithms (VQAs), in optimizing error correction based on the fidelity cost function has been explored previously, but has yielded negligible benefits compared to a system without error correction. Zoratti et al. explore the usage of an approximation of the quantum Wasserstein Distance Order 1 as the basis of a Wasserstein cost function to achieve the same purpose in their paper "Improving the speed of variational quantum algorithms for quantum error correction" [1]. They purport that a model based on this new cost function substantially improves the effectiveness of VQAs both in terms of speed and greater error correction.

1.2 Key Terms and Equations

1.2.1 Quantum Wasserstein Distance

The classical Wasserstein Distance uses the probability distributions of unit amounts of mass and a cost function based on moving a particular mass between two points [2]. The quantum Wasserstein Distance is a generalization of this with the introduction of the concept of neighbouring states. Namely, two quantum states are neighbouring if they differ by at most one qubit [3]. Then, the quantum W_1 distance is the maximum distance of the norm where each pair of neighbouring states has a distance of one. This can similarly be thought of as the Hamming distance from classical computing using bit strings, but for quantum computing.

As a result of more weight being placed in quantum states that differ greatly, the gradient in regions far from the local minima is much steeper, allowing for faster movement through barren plateaus and a higher probability of finding the optimal VQA parameters.

1.2.2 Hamiltonians

In order to compute the fidelity and Wasserstein cost functions, we need to calculate two Hamiltonians, one for each cost function. The fidelity Hamiltonian can be defined as the identity matrix over some system X minus the outer product of X's fiduciary state. In other words:

$$\hat{H}_X^{(\mathrm{fid})} := \hat{\mathbb{1}}_X - |\varnothing\rangle_X \langle\varnothing| \tag{1}$$

For our paper, we will set $|\varnothing\rangle_X$ such that it is equal to $|00\cdots0\rangle$, where the number of qubits in $|0\rangle$ is equal to the number of qubits in X.

Similarly, we can compute the Hamiltonian for the Wasserstein cost function, which is based on the quantum Wasserstein distance of order 1. It can be estimated as:

$$\hat{H}_X^{(\mathrm{wass})} := \sum_{j=1}^{n} j\hat{\Pi}_X^{(j)} \tag{2}$$

where $\hat{\Pi}_X^{(j)}$ is the sub-space of the register X, with j qubits in $|1\rangle$ and the remaining qubits in $|0\rangle$. In other words, we take the weighted sum of the outer products of states with themselves for all possible n-qubit states with 1 to n number of $|1\rangle$ qubits.

1.2.3 Cost Functions

With the Hamiltonians defined for each cost function, we can now define the actual cost function used for gradient descent. We will measure the outcomes at the end of our VQA, and use these outcomes to calculate the cost. The cost function is then defined as the expectation value of the Hamiltonian of the exact cost function we wish to use. In other words, to calculate the fidelity cost, we will take the expectation value of the fidelity Hamiltonian, and to calculate the Wasserstein cost, we will take the expectation value of the Wasserstein Hamiltonian. With the application of Born's rule, we can equivalently compute each cost respectively as:

$$C^{(\mathrm{fid})}(\vec{\alpha}, \vec{\beta}) := tr\{\hat{\rho}_X^{(V(\vec{\alpha}), W(\vec{\beta}))} \hat{H}_X^{(\mathrm{fid})}\} \tag{3}$$

and

$$C^{(\text{wass})}(\vec{\alpha},\vec{\beta}) := tr\{\hat{\rho}_X^{(V(\vec{\alpha}),W(\vec{\beta}))}\hat{H}_X^{(\text{wass})}\} \tag{4}$$

where $\hat{\rho}_X^{(V(\vec{\alpha}),W(\vec{\beta}))}$ is the density matrix taken by measuring the circuit over the system X, with the circuit performing transformations $V(\vec{\alpha})$ and $W(\vec{\beta})$ (see Fig. 1, 2, and 3 for definitions on $V(\vec{\alpha})$ and $W(\vec{\beta})$).

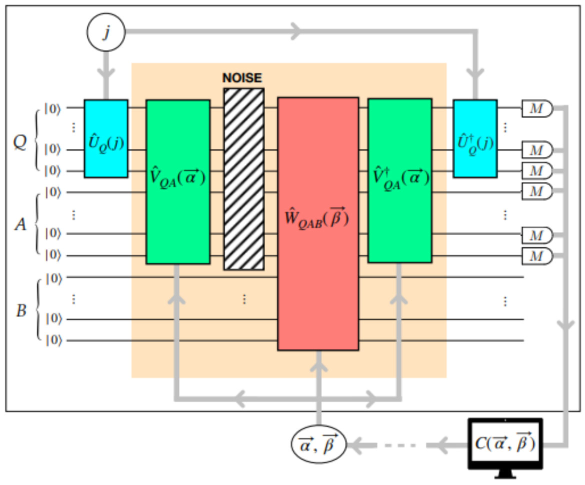
1.2.4 Fidelity Equations

The effectiveness of running the VQAs using the two different cost functions was evaluated based on the fidelity value calculated between the starting state and the state after noise recovery. The fidelity equation we used is as follows:

$$F(\rho,\sigma) = Tr(\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}})^2 \tag{5}$$

In our code, the fidelity between two states was calculated using the built-in fidelity function in PennyLane [4].

1.3 Software Implementation



j
NOISE
Q
A
B
$|0\rangle$
$\hat{U}_Q(j)$
$\hat{V}_{QA}(\vec{\alpha})$
$\hat{W}_{QAB}(\vec{\beta})$
$\hat{V}^\dagger_{QA}(\vec{\alpha})$
$\hat{U}^\dagger_Q(j)$
M
$\vec{\alpha},\vec{\beta}$
$C(\vec{\alpha},\vec{\beta})$

Figure 1: Encoding circuit module

Our overall software implementation follows Fig. 1. The details of each module are explained below.

1.3.1 Unitary 2-Set Design

To prepare Q into some desired quantum state, we need to choose a single-qubit unitary matrix. We used a set of unitaries defined by the paper. This set S consisted of the unitary matrices $\hat{\mathbb{1}}, \hat{\sigma_1}, e^{\pm\frac{i\pi}{4\hat{\sigma_1}}}, e^{\pm\frac{i\pi}{4\hat{\sigma_2}}}$ where σ_1 is the Pauli-X operator and σ_2 is the Pauli-Z operator [5]. This unitary matrix is then applied by the module $U_{Q(j)}$ in our circuit.

1.3.2 Encoding Circuit Module

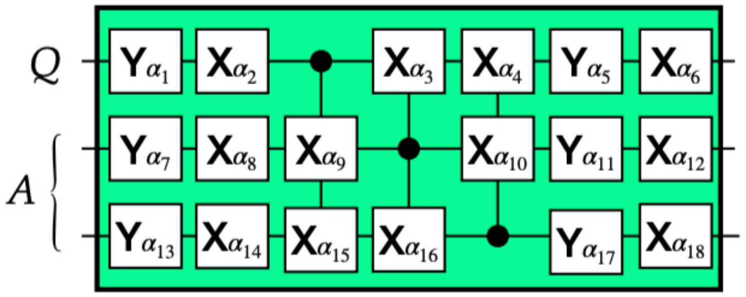

Q
A
Y_{α_1} X_{α_2} X_{α_3} X_{α_4} Y_{α_5} X_{α_6}
Y_{α_7} X_{α_8} X_{α_9} $X_{\alpha_{10}}$ $Y_{\alpha_{11}}$ $X_{\alpha_{12}}$
$Y_{\alpha_{13}}$ $X_{\alpha_{14}}$ $X_{\alpha_{15}}$ $X_{\alpha_{16}}$ $Y_{\alpha_{17}}$ $X_{\alpha_{18}}$

Figure 2: Encoding circuit module, labelled $V_{QA}(\vec{\alpha})$

The module V_{QA} is responsible for distributing the information in Q over the larger system QA before it is subjected to noise. It performs multiple Pauli-X/Y rotations, based on the values given by α. Note that later in this circuit, this transformation is undone by applying the adjoint, to recover the information that was in QA into Q.

1.3.3 Noise Correction Module

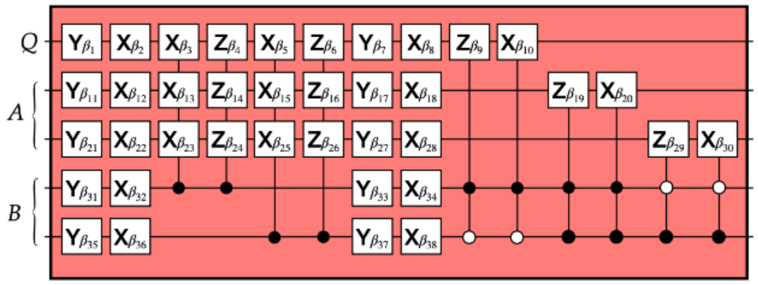

Q
A
B
Y_{β_1} X_{β_2} X_{β_3} Z_{β_4} X_{β_5} Z_{β_6} Y_{β_7} X_{β_8} Z_{β_9} $X_{\beta_{10}}$
$Y_{\beta_{11}}$ $X_{\beta_{12}}$ $X_{\beta_{13}}$ $Z_{\beta_{14}}$ $X_{\beta_{15}}$ $Z_{\beta_{16}}$ $Y_{\beta_{17}}$ $X_{\beta_{18}}$ $Z_{\beta_{19}}$ $X_{\beta_{20}}$
$Y_{\beta_{21}}$ $X_{\beta_{22}}$ $X_{\beta_{23}}$ $Z_{\beta_{24}}$ $X_{\beta_{25}}$ $Z_{\beta_{26}}$ $Y_{\beta_{27}}$ $X_{\beta_{28}}$ $Z_{\beta_{29}}$ $X_{\beta_{30}}$
$Y_{\beta_{31}}$ $X_{\beta_{32}}$ $Y_{\beta_{33}}$ $X_{\beta_{34}}$
$Y_{\beta_{35}}$ $X_{\beta_{36}}$ $Y_{\beta_{37}}$ $X_{\beta_{38}}$

Figure 3: Noise correction module, labelled $W_{QAB}(\vec{\beta})$

The module W_{QAB} is responsible for the actual error correction after the noise has been applied to QA. Similar to V_{QA}, it performs multiple Pauli-X/Y/Z rotations, based on the values given by β. It uses an additional quantum register B to facilitate this operation.

1.3.4 Gradient Descent

The alpha and beta parameters were optimized through the use of gradient descent with momentum, as outlined by the paper. The Hamiltonians for the cost functions were applied over QA, meaning we have $\hat{H}_{QA}^{(\text{fid})}$ for fidelity cost and $\hat{H}_{QA}^{(\text{wass})}$ for Wasserstein cost. The original paper outlined equations for computing the gradient of each parameter, however we opted to use PennyLane's *step_and_cost* function that is a part of the *MomentumOptimizer* class [6]. Using PennyLane's built in functions reduced the time required to reproduce the results as well as the overall complexity of the code. The *step_and_cost* function takes as input the appropriate VQA cost function to use and the alpha and beta parameters. Each call returned a resulting cost and the updated alpha and beta parameters based on the gradients and momentum. Gradient descent was run until either the gradient converged or the maximum number of iterations was reached. We defined convergence as when the difference between the current cost and the last cost is less than 10^{-4}.

1.3.5 Noise Simulation Kraus Operators

For the noise model in our circuit, we focused on simulating two different types of noise: bit flip noise, and phase flip noise. These noise models are defined by a set of Kraus operators, one for each noise model, and is represented by the set

$$\{\hat{K}_{QA}^{(0)} \cdots \hat{K}_{QA}^{(l)} \cdots \hat{K}_{QA}^{(n)}\} \tag{6}$$

where n is the number of qubits in QA, and

$$\hat{K}_{QA}^{(0)} = \sqrt{1-p}\ \hat{\mathbb{1}}_{QA} \tag{7}$$

and

$$\hat{K}_{QA}^{(l)} = \sqrt{\frac{p}{n}}\ \hat{\sigma}^{(l)}. \tag{8}$$

$\hat{\sigma}^{(l)}$ is defined as the Pauli operator applied to the lth qubit that defines the noise species. In our case, phase flip would be applying the Pauli-Z operator, and bit flip would be applying the Pauli-X operator. The noise was applied over the QA registers, and since QA is 3 qubits

(1 qubit for Q and 2 qubits for A, defined by the paper), each set of Kraus operators would have 4 total Kraus operators.

For our software implementation, we represented $\hat{\sigma}^{(l)}$ as the matrix for the Pauli operator applied to the lth qubit, so for example, bit flip noise on the 2nd qubit would be $\hat{\sigma}^{(2)} = \mathbb{1} \otimes X \otimes \mathbb{1}$, where $\mathbb{1}$ is the 2x2 identity matrix, and X is the 2x2 Pauli X matrix. Finally, these Kraus operators can be used to form a set which we then pass to *qml.QubitChannel* to perform the actual noise modelling in our circuit.

We originally attempted to use library functions such as *qml.BitFlip* [7] and *qml.PhaseFlip* [8], however the Kraus operators that are applied are slightly different, with Pennylane defining $\hat{K}_{QA}^{(l)}$ as $\sqrt{p}\ \hat{\sigma}^{(l)}$ for only up to $l = 1$ wire. Applying this to each wire would yield different results compared to the Kraus operators used in the original paper.

1.3.6 Simulation Design

A single simulation is constructed as follows. A simulation is used for one cost function and one noise model and iterates over all the possible unitaries in the 2-Set Design. For each unitary we initialize the alpha and beta parameters randomly and then perform gradient descent for a set number of maximum iterations or until convergence is reached. After gradient descent, the final density matrix is found for the respective final alpha and beta parameters applied to the circuit. Once all of the unitaries have been used, all the density matrices are summed and divided by the total number of density matrices. This average density matrix is used alongside the initial density matrix to calculate the average fidelity of that simulation.

2 Key Results

The paper had a variety of results for both bit flip noise and phase flip noise. While our implementation was not an exact replication of the results found in the paper due to limitations in time and hardware, we believe that our implementation does follow the same trends and strongly supports the findings of Zoratti et al (2023).

The core results presented in the paper can be summarized into the following three points:

- The Wasserstein cost function has a higher percentage of successful runs compared to the fidelity cost function.
- The Wasserstein cost function reaches convergence much faster than the fidelity cost function.
- Running Wassertein cost simulations following fidelity simulations yields further improvements to cost, but the same cannot be said about fidelity then Wasserstein.

In the sections below, you can find figures and explanations comparing our results to ones found in the original paper.

Before we get into the three core results, we were first able to replicate the value of 0.822 for F_0 which the authors claimed to be the fidelity value in a system without noise correction. After successfully replicating this result, we used this F_0 value as a benchmark for defining whether a particular simulation is successful.

All results were run with the following settings.

- 80 simulations
- 300 iterations per unitary = 1800 iterations per simulation
- 80% noise probability
- 0.01 step size
- 0.9 momentum

2.1 Bit Flip Noise Results

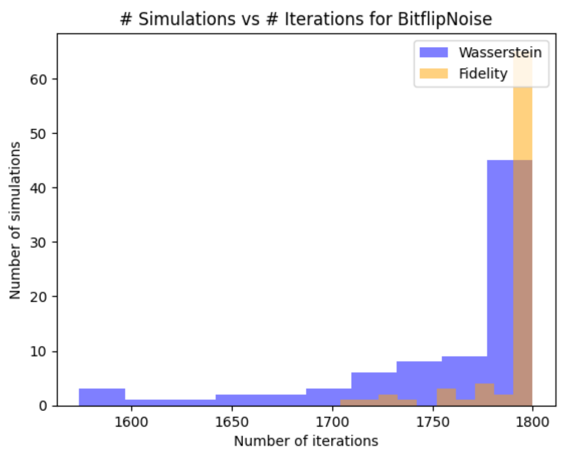


Figure 4: The number of iterations for cost function convergence using the bit flip noise model

For Fig. 4, we were able to simulate the experiment where the authors examined how many simulations converged early for the Wasserstein cost function compared to the fidelity cost function. As we can see, the majority of the fidelity simulations took 1800 iterations where as many Wasserstein simulations were able to converge within 1500-1700 iterations.

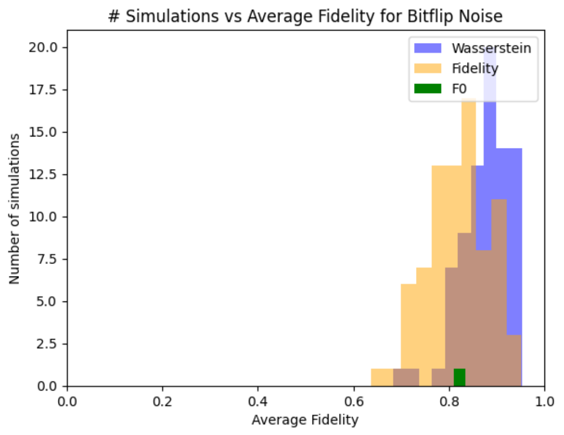


Figure 5: The resulting average fidelities using the bit flip noise model

Similarly, for Fig. 5, we see that on average, the Wassterstein cost function simulations usually result in a better fidelity value between the starting and ending state when compared to the fidelity cost function simulations. This is in line with the conclusions presented in the paper for the bit flip noise model.

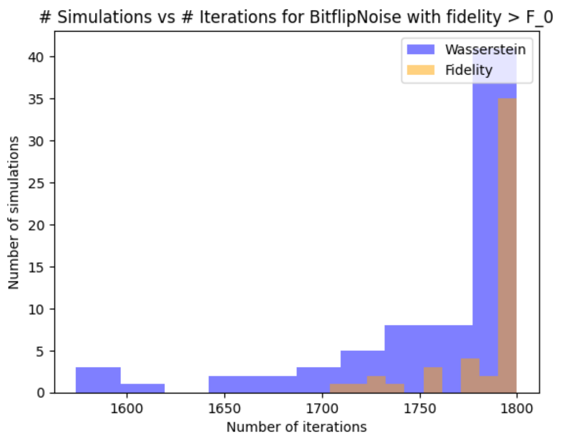


Figure 6: The number of iterations for cost convergence using the bit flip noise model filtered only for simulations that had a better fidelity than F_0

Fig. 6 is the same figure as Fig. 4 except that we filter out simulations where the fidelity was not above F_0. In other words, only data from successful runs are plotted in this figure. This graph is more representative and closer to the graph that was displayed in the original paper and additionally provides evidence that Wasserstein simulations converge faster than fidelity simulations.

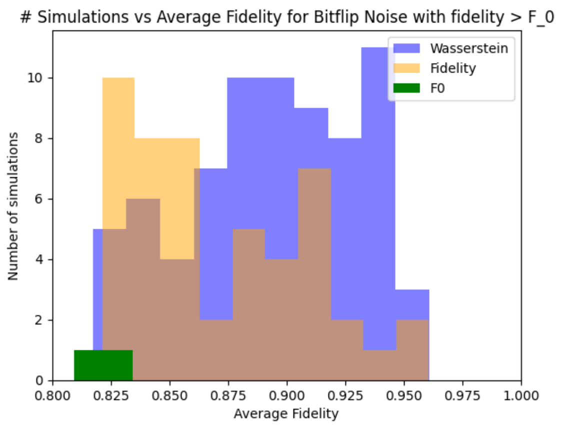


Figure 7: The resulting average fidelities using the bit flip noise model filtered only for simulations that had a better fidelity than F_0

Fig. 7 is the same as Fig. 5 except we also filter out simulations where the fidelity was not above F_0. Here we see that unlike the paper, some of the fidelity simulations gave us a fidelity value above F_0, but the number of simulations above F_0 was still less for the fidelity cost function than for Wasserstein.

2.2 Phase Flip Noise Results

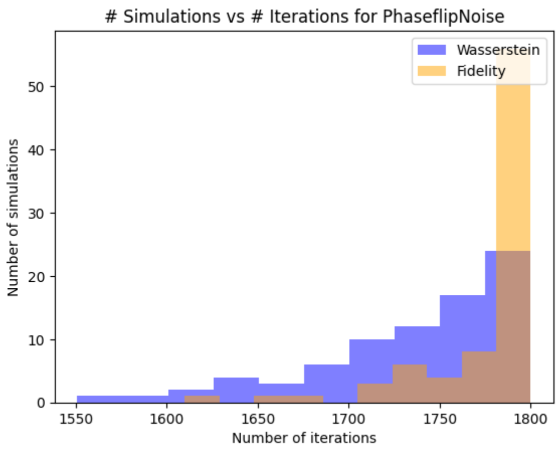


Figure 8: The number of iterations for cost convergence using the phase flip noise model

In Fig. 8, we perform the same experiment as in Fig. 4, using the phase flip noise model instead of bit flip. We can see that we have similar results as Fig. 4, where the majority of simulations run using the fidelity cost function use all 1800 iterations, as opposed to simulations run using the Wasserstein cost function, which tended to have more simulations that converged sooner. The number of simulations that converged sooner were higher for both fidelity and Wasserstein cost functions for this noise model compared when we applied bit flip noise.

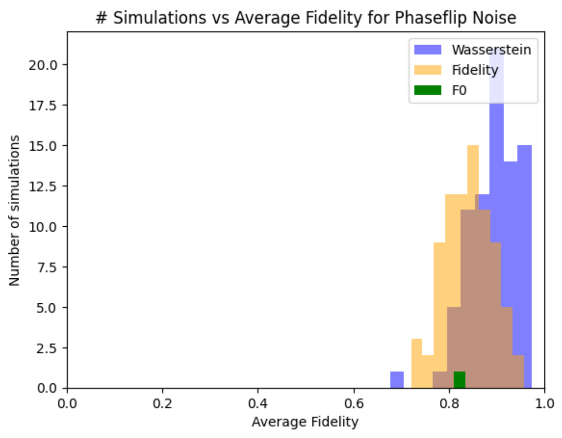


Figure 9: The resulting average fidelities using the phase flip noise model

Similarly, in Fig. 9, we have Wasserstein simulations performing better than fidelity simulations in terms of average fidelity, with the majority of Wasserstein simulations having fidelities exceeding the threshold F_0. However, there are still a substantial number of simulations run using the fidelity cost function that also result in fidelities exceeding F_0. This is slightly different from the findings in the paper, however the conclusion that Wasserstein outperforms fidelity cost still holds.

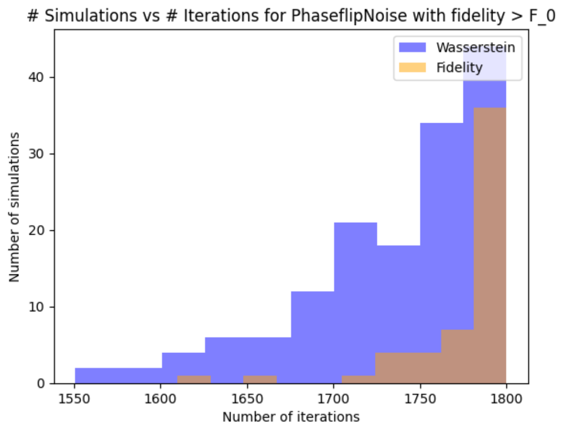


Figure 10: The number of iterations for cost function convergence using the phase flip noise model filtered only for simulations that had a better fidelity than F_0

In Fig. 10 we have the same results as Fig. 8, however we filter out simulations where the fidelity did not exceed the threshold F_0. This graph is more representative and closer to the graph presented in the original paper, and further shows that the Wasserstein cost function model converges faster, and provides better results compared to using the fidelity cost function.

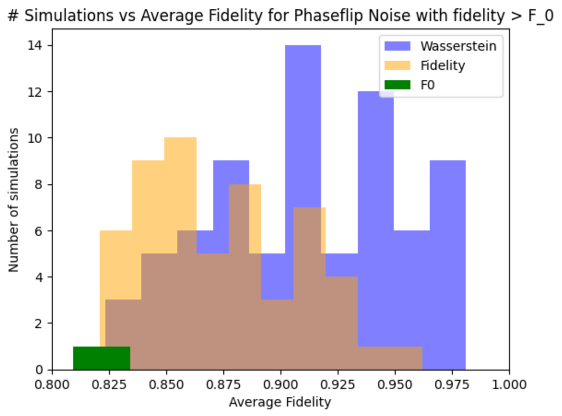


Figure 11: The resulting average fidelities using the phase flip noise model filtered only for simulations that had a better fidelity than F_0

In Fig. 11, we have the same results as Fig. 9, however we filter out simulations where the fidelity did not exceed the threshold F_0. This graph is more representative and closer to the graph presented in the original paper, but similar to Fig. 9, we still have a sizeable number of simulations run using the fidelity cost function that exceed F_0, unlike the original paper. However, we still see that using the Wasserstein cost function yields higher average fidelities than using fidelity cost.

2.3 Consecutive Cost Functions Results

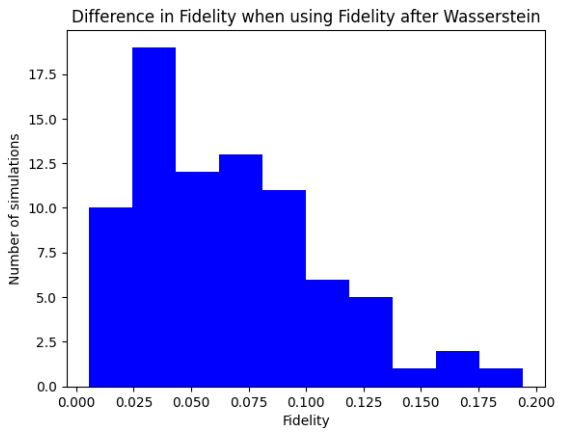


Figure 12: The improvement in fidelity when using the fidelity cost function on the parameters determined by Wasserstein

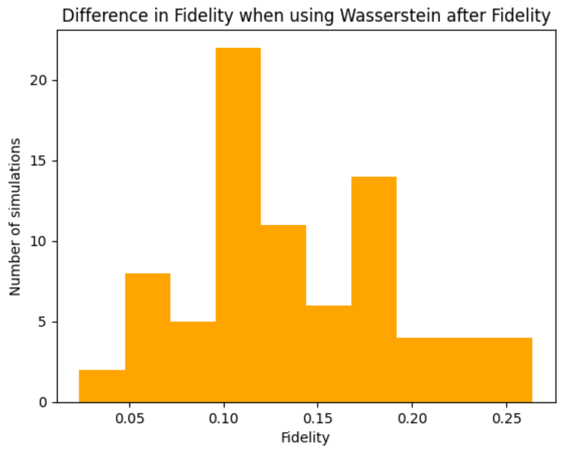


Figure 13: The improvement in fidelity when using the Wasserstein cost function on the parameters determined by fidelity

An additional set of results was run where one cost function was first used for gradient descent followed by a second cost function which used the alpha and beta parameters optimized by the first cost function to further perform gradient descent. Finally the difference between the fidelity of the output of the first cost function and the second cost function was compared. This helps visualize how much each cost function could improve on the values outputted by the other. In Fig. 12 and Fig. 13 we can see that the fidelity cost function does not improve on the fidelity as much as the Wasserstein cost function does. The majority of Fig. 12 only shows an improvement between 0.00 to 0.10 in fidelity while Fig. 13 shows an improvement between 0.05 to 0.20. This is in line with what the conclusions of the paper, where the Wasserstein cost function was able to provide much further improvements than the fidelity cost function.

3 Rationale for Framework

There were a number of other quantum frameworks our team considered like Qiskit [9], Strawberry Fields [10], QuTip [11] and Cirq [12] prior to settling on PennyLane as our framework of choice. One major factor in selecting PennyLane versus any of the other quantum frameworks was our team's familiarity with the framework as all of our assignments and in-class demos were completed using PennyLane. Given our previous exposure to PennyLane, we predicted that using PennyLane would reduce the amount of time we would otherwise need to spend learning one of the other quantum computing frameworks. The second reason our team chose the PennyLane framework compared to frameworks like Cirq and Qutip is that PennyLane's momentum gradient descent optimizers provide automatic differentiation meaning we would not be required to manually calculate the gradient for back propagation. Additionally, PennyLane allows us to implement custom noise channels, which some of the other frameworks do not have support for. This is an important point because the authors define their own multi-qubit Kraus operators as mentioned above and to accurately reproduce the noise, we needed to have the ability to create a custom noise channel. Finally, PennyLane is known to be optimized for hybrid quantum computer and classical computer algorithms which is the type of algorithm that was implemented by our paper and as such, we believed that PennyLane would also provide us the best performance on this front.

4 Reproducibility

There were a number of issues that our team encountered when trying to reproduce our paper's results. The first big issue that our team found was that neither the type of momentum used for gradient descent nor the momentum hyperparameters were mentioned. As a result, our team experimented with many different hyperparameter values before settling on one that gave us results most similar to the ones seen in the paper. Additionally, our team made the assumption that the paper's implementation used regular gradient descent with momentum instead of one of the momentum variations like Nesterov or Adam.

Understanding which operations to perform in a single simulation proved to be a challenge for reproducing the results. The paper lacked information on how the alpha and beta parameters were randomly initialized at the start of each simulation and how the set of unitaries was sampled for each simulation. Additionally, the authors of the paper never mentioned their definition for cost convergence and how this was calculated.

To overcome this missing information in the paper, our team made a number of reasonable assumptions. The first assumption we made was that the authors sampled alpha and beta parameters from a uniform random distribution. The second assumption that our team made was that the alpha and beta parameters were optimized for a single unitary in the set, before moving onto the next unitary. While the authors listed the number of iterations per simulation, it was unclear how the iterations were divided amongst all the unitary matrices in the set and as a result, we divided the number of iterations per unitary evenly. A different alternative our team also considered was to randomly select a unitary matrix from the set for each iteration before optimizing alpha and beta to avoid overfitting the alpha and beta values to a single unitary in the set. Lastly, our team experimented with many different values and definitions for cost convergence before settling on 10^{-4} as our convergence value since this value yielded results that were most similar to the paper.

Another issue we discovered while implementing the paper was that the F_0 value for bit flip noise that we produced was different than the one in the paper. For bit flip noise, we found that $F_0 \approx 0.288$, whereas the authors had $F_0 \approx 0.822$. The assumption we made here was that the authors simply used the F_0 value from the phase flip noise circuit which was in fact 0.822 and one of the results that we were able to reproduce as seen above.

We also encountered practical limitations with the runtime of the simulations to reproduce results. Running our results for one noise model and one cost function took slightly under 4 hours even on our best hardware. The extensive amount of time it took to complete each set of simulations made it difficult to iterate on our results.

5 Conclusion

Through this project, our team validated the results shown by the authors Zoratti et al. in their paper "Improving the speed of variational quantum algorithms for quantum error correction". Namely, we tested the performance of a variational quantum algorithm using two cost functions: fidelity and the approximation of the quantum Wasserstein distance of order 1. We compared the performance of these two cost functions with respect to the speed of convergence and quantity of successful runs and similarly concluded that the Wasserstein cost function outperforms the fidelity cost function on both fronts. The key difference between the Wasserstein and fidelity cost functions is that the latter is greatly impacted by the phenomenon of barren plateaus, leading to slow or no convergence at all. The importance of selecting a cost function that is able to move through barren plateaus is so critical that the inability of the fidelity cost function to do so resulted in it yielding similar results to a system with no error correction at all.

The simulations that we ran were only for the single qubit case (ie. Q is a single qubit) but further examination of how this generalizes to the multi-qubit case should be explored. The impact of different types of momentum on the phenomenon of barren plateaus as well as different values for the momentum and stepsize hyperparameters can also be investigated to determine optimal parameter values.

6 Reference List

[1] Zoratti, F., De Palma, G., Kiani, B., Nguyen, Q. T., Marvian, M., Lloyd, S., amp; Giovannetti, V. (2023, March 27). Improving the speed of variational quantum algorithms for quantum error correction. arXiv.org. Retrieved February 9, 2023, from https://arxiv.org/abs/2301.05273

[2] De Palma, G. (2023, April). The quantum Wasserstein distance of order 1. Lecture.

[3] De Palma, G., Marvian, M., Trevisan, D., amp; Lloyd, S. (n.d.). The quantum Wasserstein distance of order 1. Indico Physik. Retrieved April 7, 2023, from https://indico.physik.uni-muenchen.de/event/84/attachments/247/493/S1A.DePalma.abstract.pdf

[4] Qml.math.fidelity - pennylane. qml.math.fidelity - PennyLane 0.29.1 documentation. (n.d.). Retrieved March 28, 2023, from https://docs.pennylane.ai/en/stable/code/api/ pennylane.math.fidelity.html

[5] Di Matteo, O. (n.d.). A short introduction to unitary 2-designs. Retrieved April 1, 2023, from https://glassnotes.github.io/OliviaDiMatteo-Unitary2Designs.pdf

[6] Qml.MomentumOptimizer - Pennylane. qml.MomentumOptimizer - PennyLane 0.29.1 documentation. (n.d.). Retrieved March 29, 2023, from https://docs.pennylane.ai/en/stable/code/api/pennylane.MomentumOptimizer.html

[7] Qml.BitFlip - Pennylane. qml.BitFlip - PennyLane 0.29.1 documentation. (n.d.). Retrieved March 20, 2023, from https://docs.pennylane.ai/en/stable/code/api/pennylane .BitFlip.html

[8] Qml.PhaseFlip - Pennylane. qml.PhaseFlip - PennyLane 0.29.1 documentation. (n.d.). Retrieved March 20, 2023, from https://docs.pennylane.ai/en/stable/code/api/pennylane.PhaseFlip.html

[9] Qiskit Overview. Qiskit.org. (n.d.). Retrieved April 7, 2023, from https://qiskit.org/overview/

[10] Strawberry Fields. (n.d.). Retrieved April 7, 2023, from https://strawberryfields.ai/

[11] Nation, P. D., amp; Johansson, J. R. (n.d.). Qutip Quantum Toolbox in Python. Retrieved April 7, 2023, from https://qutip.org/features.html

[12] Cirq. Google Quantum AI. (n.d.). Retrieved April 7, 2023, from https://quantumai .google/cirq

[13] Di Matteo, O. (n.d.). Understanding the Haar measure. Understanding the Haar measure - PennyLane documentation. Retrieved April 1, 2023, from https://pennylane.ai/qml/demos/tutorial_haar_measure.html

[14] Di Matteo, O. (n.d.). Unitary Designs. Unitary designs - PennyLane documentation.

Retrieved April 1, 2023, from https://pennylane.ai/qml/demos/tutorial_unitary_designs.html