Project No: A3099-191

NANYANG TECHNOLOGICAL UNIVERSITY

Quantum Autoencoder

Submitted by: Zhang Zeyu

Matriculation Number: U1620474K

Supervisor: Jiang Xudong

Co-supervisor: Kwek Leong Chuan

School of Electrical & Electronic Engineering

A final year project report presented to the Nanyang Technological University in partial fulfilment of the requirements of the degree of Bachelor of Engineering

2020

Table of Contents

Abstract

Quantum information and machine learning are two highly active research fields in the modern scientific community. In the thesis, we study some of the research results born from the fusion of these two branches-quantum machine learning, discuss about the search methodology they adopted, explore the untouched area in the topics and provide our own insights towards the problem. More specifically, we first proposed an alternative problem-solving technique for quantum data compression by doing numerical constrained optimization on a classical computer instead of running gradient descent technique on a set of physical apparatus in the original work [1]. Next, the viability of our proposed techniques and its relation to another exiting data compression technique: principal component analysis are discussed in terms of possible applications. Finally, on the topic of denoising corrupted data, the proposed quantum denoising autoencoder is studied and its generalizability is explored by testing the performances a several quantum states and with different types of noises which were not mentioned by the original work [2].

Acknowledgements

The author would like to take this opportunity to express his deepest gratitude to his supervisor, Professor Jiang Xudong, for supervising the author for this final-year project and the advice for enhancing the project tasks. The author had many inspirations drawn from the discussions with Professor Jiang for making topics more coherent and complete.

The author would like to extend his heartiest gratitude to his co-supervisor, Professor Kwek Leong Chuan, for spending time sorting out the detailed schedule and giving constructive feedbacks on all related experiments. The author has learned much from Professor Kwek, both in the knowledge on the relevant topics and the attitude towards research throughout the whole journey of doing the final-year project for the past ten months.

Zhang Zeyu

May 2020

Acronyms

CQAE	Compressive Quantum Autoencoder
CQML	Classical-Quantum Machine Learning
DAE	Denoising Autoencoder
GHZ state	Greenberger–Horne–Zeilinger state
PCA	Principal component analysis
QCML	Quantum-Classical Machine Learning
QDAE	Quantum Denoising Autoencoder
QNN	Quantum Neural Network(s)
QQML	Quantum-Quantum Machine Learning

Symbols

$\|\psi\rangle$	wavefunction in $meter^{-\frac{1}{2}}$ (1-dimentional)
$\|\mathrm{GHZ}\rangle$	Greenberger–Horne–Zeilinger state (GHZ state)
$\|\mathrm{GHZ}\rangle^{\mathrm{d}}$	d-dimensional GHZ state (general case)
$\|\mathrm{GHZ}\rangle^{\mathrm{M}}$	M-partite qubit GHZ state (2-dimensional subsystem case)
$\|\mathrm{W}\rangle^{\mathrm{d}}$	d-dimensional W state
$\|+\rangle^{\mathrm{n}}$	Plus state formed by n qubit Plus state
$\|0\rangle^{\mathrm{n}}$	Zero state formed by n qubit Zero state
σ_x	Pauli x operator
σ_y	Pauli y operator
σ_z	Pauli z operator
C	cost function (unitless)
L	objective/loss function (unitless)
F	fidelity (unitless)
ψ	noisy quantum state
ϕ	noiseless state
U	unitary operator/matrix
ρ	density matrix
λ_x	eigenvalues of ρ
$\otimes$	tensor product
$I^{\otimes \mathrm{i}}$	repeated tensor product for i times on identity matrix
$S(\rho)$	Von Neumann entropy of density matrix ρ
$H(X)$	Shannon entropy of the associated with probability distribution of random variable X
R^{n}	real vector space with dimension n
C^{n}	complex vector space with dimension n

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter the overview of the final-year project will be presented. This comprises of the following elements: the motivations of doing this project, the scope of the main tasks and the objective to be achieved, and the generic structure to approach various subtasks encountered in the study.

1.1 Motivations

Machine learning has been a rapidly growing field in computer science and engineering in the last decade, with steady progress in image recognition, computer vision, voice recognition, medical diagnosis, search engines and so forth. Machine learning algorithms falls under the following categories: supervised training, unsupervised training and reinforcement learning. Recently, there has been intensive theoretical and in-principle experimental proofs of applying quantum mechanics to machine learning. This furry of activities has been fueled in part by the promise of quantum technologies to provide some advantage over the classical counterparts. One example is the research on quantum autoencoder uses machine learning to represent quantum data in a lower dimensional space, in short, quantum autoencoders compress quantum data so as to reduce memory requirements.

Compression of quantum states could have vast potential applications. One such example would be in quantum communication and quantum networking field. Transmission of quantum states is not a trivial task as one needs to fight decoherence and noise that threat the fidelity of the information of interest. On other hand, it costs much less recourses to keep the coherence of the quantum states if there is a

representation of the states in lower dimension. Therefore, there are growing interest to use machine learning technique to learn such a representation under certain conditions. The related field of studies are quantum information theory, machine learning and constrained optimization.

On the other hand, quantum states are often corrupted by different types of noises in the practical situation. Hence, it is essential to be able to get rid of the noises before they can be used in various applications. From economical point of view, denoising quantum states incurs much lower cost than preparing new clean quantum states; from technical point of view, denoising multiple quantum states is a less challenging task than preparing new clean quantum states. These two points will be elaborated in Section 2.4.9.

1.2 Objectives and Scope

Two experiments related to autoencoder were investigated as part of the project. In 2019, Pepper et al. has considered a simple experimental realization of an autoencoder to reduces qutrits to qubits with low error rates. In this project, we will follow closely an experimental realization of a quantum autoencoder with qutrits [1] or more generally with a compression of qudits to qunits (d >n). One possible alternative approach of the original method is proposed. This technique potentially could be extended to applications of machine learning to other possible quantum information tasks. A perspective from information theory will be adapted for investigation as well.

The second work by D. Bondarenko and P. Feldmann on denoising quantum encoder (QDAE) [2] showed how was its performance on the Greenberger–Horne–Zeilinger states. We explore QDAE's performance further by testing the QDAE with other useful quantum states: W states, Zero states and Plus state. The generalizability of QDAE is examined by testing its denoising performances on states and/or noises unseen by it during the training phase. This is one important metric to gauge if QDAE

has the potential to be applied for more diverse and complicated quantum machine tasks.

1.3 Organisations

The entire thesis is divided into the following sections:

First, some of the concepts pertinent to our study are presented. Next, three research papers are reviewed. Original ideas are built on top of those works and expanded into problem statements, formulations and designs of experiments. Afterwards, elaborate experiments and analyses are conducted. From the results of the experiments, we eventually give discussions about the new insights gained, problems left unsolved and more potential work could be done for the further investigation.

Chapter 2

Literature Review

2.1 Review of Definitions

2.1.1 Quantum Information

- An overview of quantum information

  In physics and computer science fields, quantum information corresponds to the state of a quantum system. It is quantum information theory's basic study element [3]. Quantum information processing methods can be used to. Quantum Information can be processed with computers, transferred from location to location, controlled with algorithms, and examined with mathematics.

- The domain of quantum machine learning [4]

  Classical-Quantum Machine Learning (CQML): exploit quantum algorithms to speed up classical machine learning tasks
  Quantum-Classical Machine Learning (QCML): exploit classical machine learning techniques to improve quantum tasks
  Quantum-Quantum Machine Learning (QQML): exploit quantum computing devices to carry out learning tasks with quantum data

- Bra-ket (Dirac) Notation: a shorthand notation used to represent a quantum state, introduced by physicist Paul Dirac and is commonly adopted in quantum physics community (e.g. $|0\rangle$, $|1\rangle$ where they are also called the basis/computational state).

- State vector: a vector that represents a certain quantum state. A state vector must have norm one according to the law of quantum mechanics. It is often denoted by $|\psi\rangle$. Two common two-dimensional state vectors are:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

- Quantum superposition: a state vector be formed by linear combination (as defined in linear algebra) of several basis/computational states as long as it is normalized. For example,

$$|\psi\rangle = x_0|0\rangle + x_1|1\rangle + x_2|2\rangle + \cdots x_n|n\rangle$$

- Qubit: fundamental unit of quantum information, the counterpart of bit in classical computing. A qubit exists in superposition (linear combination) of two basis/computational states. For example,

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{bmatrix}$$

- Qutrit: a qutrit is a unit of quantum state which exists in superposition (linear combination) of three basis/computational states. For example,

$$|\psi\rangle = \frac{1}{\sqrt{3}}(|0\rangle + |1\rangle \ + |2\rangle) = \frac{1}{\sqrt{3}}\left(\begin{bmatrix}1\\0\\0\end{bmatrix} + \begin{bmatrix}0\\1\\0\end{bmatrix} \ + \begin{bmatrix}0\\0\\1\end{bmatrix}\right) = \begin{bmatrix}\frac{1}{\sqrt{3}}\\ \frac{1}{\sqrt{3}}\\ \frac{1}{\sqrt{3}}\end{bmatrix}$$

- Qudit: generalization of qubit, can exist in superposition (linear combination) of arbitrary number of states ((e.g. $|0\rangle$, $|1\rangle$, ..., $|d\rangle$).

- Qunit: same as qudit. ‘n’ and ‘d’ represent some arbitrary states.

- Entanglement: in the context of quantum physics and quantum mechanics, entanglement is a special property shared by more than one quantum object. The entangled objects are said to be in the entangled state, and they do not have individual physical properties. When one object is observed, all the rest objects’ properties are known at the same time. Entanglement is a quantum mechanical property and does not exist in classical physics.

- Tensor product: in the context of quantum computing, tensor product is a mathematical operator to join individual sub-quantum system for form a larger quantum system and it is denoted by $\otimes$. For instance,

$$|0\rangle \otimes |0\rangle = |00\rangle$$

- Greenberger–Horne–Zeilinger state (GHZ state): a type of quantum state that exhibits entanglement. It assumes the form

$$|\mathrm{GHZ}\rangle^{\mathrm{d}} = \frac{1}{\sqrt{d}} \sum_{i=0}^{d-1} |i\rangle \otimes \ldots \otimes |i\rangle = \frac{1}{\sqrt{d}} (|0\rangle \otimes \ldots \otimes |0\rangle + \cdots + |d-1\rangle \otimes \ldots \otimes |d-1\rangle)$$

In the case of each of the subsystems being two-dimensional, that is for qubits, it reads:

$$|\mathrm{GHZ}\rangle^{\mathrm{M}} = \frac{|0\rangle^{\otimes \mathrm{M}} + |1\rangle^{\otimes \mathrm{M}}}{\sqrt{2}}$$

where M is some positive integer and $|\psi\rangle^{\otimes \mathrm{M}}$ denotes M number of $|\psi\rangle$ being joined together by tensor product to form a quantum state which occupies a space M times larger than the space occupied by $|\psi\rangle$.

- W state: another type of quantum state that exhibits entanglement. It assumes the form

$$|\mathrm{W}\rangle^{\mathrm{d}} = \frac{1}{\sqrt{d}}(|100\ldots0\rangle + |010\ldots0\rangle + \cdots + |00\ldots01\rangle)$$

where d is the total number of superposed (linearly combined) states.

- Plus state: one of the two basis states in the so-called 'complementary bases. Plus state $|+\rangle^{\mathrm{n}}$ is defined as the following:

$$|+\rangle^{\mathrm{n}} = \frac{1}{\sqrt{n}}(|0\rangle + |1\rangle + \cdots + |n\rangle) = \begin{bmatrix} \frac{1}{\sqrt{n}} \\ \vdots \\ \frac{1}{\sqrt{n}} \end{bmatrix} = |+\rangle^{\otimes \mathrm{n}} = |+\rangle \otimes |+\rangle \ldots \otimes |+\rangle$$

When $n = 2$, case (most common case),

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\left(\begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

- Zero state: generalized version of the qubit zero state $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$:

$$|0\rangle^{\mathrm{n}} = |0\rangle^{\otimes \mathrm{n}} = |0\rangle \otimes |0\rangle \otimes \ldots \otimes |0\rangle = |0\ldots0\rangle = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}$$

- Pure state: quantum state that can be represented by a single state vector.

- Mixed state: quantum state that cannot be represented by a single state vector. It is usually because classical probability is involved in preparation of such state.

- Density matrix: a density matrix completely describes a quantum mechanical system. While a state vector can only describe pure state, density matrix can describe mixed state (and by extension pure state). It is defined as

$$\rho \equiv \sum_i p_i \, |\psi_i\rangle\langle\psi_i|$$

where $|\psi_i\rangle$ is the i-th input state vector and $\langle\psi_i|$ is its conjugate transpose. p_i is the probability of occurring of i-th input state vector (for example the ratio of the total number of $|\psi_i\rangle$ to the total number of all the states).

- Partial trace: a mathematical operation to find the density matrix of a sub-quantum system. To illustrate, if a quantum system is form jointly by particle A and B, when using partial trace to trace out A, the result is the density matrix of B.

- Quantum logic Gate: quantum analogs of classical logic gates. They execute different types of unitary operations on quantum states and transform them into other quantum states.

2.1.2 Classical Machine Learning

- Artificial neural network

  Artificial neural networks (ANN) are computing systems that resemble biological neural networks what animal brains are composed of. However, they not identical. Such systems "learn' to perform tasks by training through a large number of examples in order to find out certain pattern from it. The artificial neutral network generally is programmed with task-specific rules.

- Constrained optimization

  The process of objective function optimization in respect of some constrained variables is known as constrained optimization. For this process, it aims to minimize the loss or energy function, and to maximize the reward or utility function which are the potential candidates of objective function. Constraints can be hard constraints or soft constraints. Hard constraints set conditions for the variables that are to be satisfied. If the conditions on the variables are not satisfied, variable values of soft constraints are penalized in the objective function.

- Gradient descent

  Gradient descent is a first-order recursive optimization algorithm for locating the minima of a concave or convex function. It was initially proposed by Cauchy in 1847 [5][6]. At each point in the iteration, steps proportionate to the negative of the gradient (or approximate gradient) of the function should be chosen to compute a local minimum of a function. If steps proportionate to the positive of the gradient are taken, a local maximum of that function is approached instead. Such a protocol is termed as gradient ascent.

- Autoencoder

  An autoencoder is one form of artificial neural network. It is adapted to find out faithful data compressions in an unsupervised setting, typically in terms of dimensionality reduction [7]. The target of an autoencoder is to learn a representation (encoding) for a set of data. This could be done by training the network to ignore unwanted data components. A reconstructing phase is learnt along with the reduction phase. For this stage, the autoencoder tries to generate a representation as close as possible to its original input from the reduced encoding. There are several variants being proposed aside the basic model. They have a unanimous goal of assigning the learned representations of the input with presumably meaningful attributes [8]. Some examples of autoencoder include the regularized autoencoders (proven effective in learning representations for relevant classification tasks) [9] and variational autoencoders (with their recent applications as generative models) [10]. Autoencoders are effective in solving many real-world tasks, ranging from understanding the semantic meaning of sentences to face recognition [11, 12].

2.2 Compressive Quantum Autoencoder (CQAE)

The work of Pepper et al. demonstrated how to use an optical setup to find more concise representations of certain quantum states with the similar rationale of an autoencoder [1].

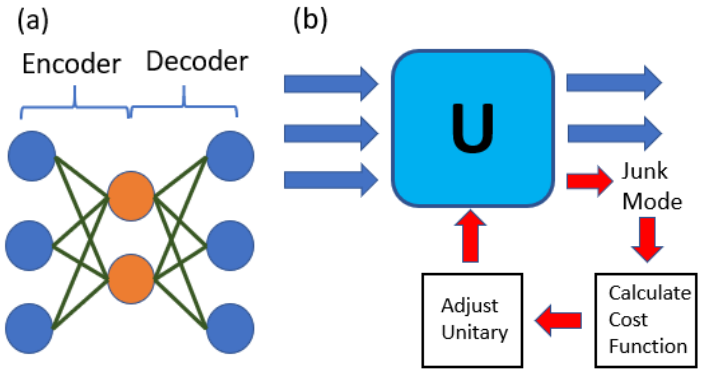

Fig. 2.1(a) Illustration of the structure of the CQAE (the encoder part) (b) Illustration of the training protocol of the CQAE (adapted from [1])

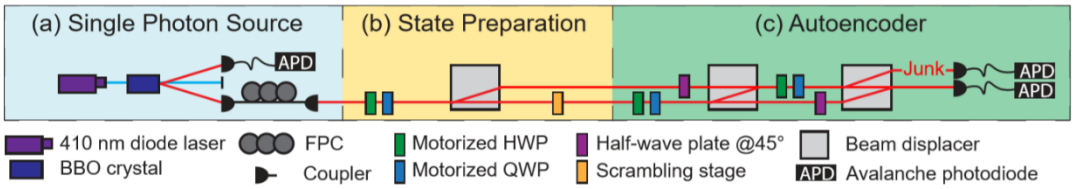

Fig. 2.2 Illustration of the physical setup of the CQAE (cited from [1])

The entire setup is said to be equivalent to the encoder part of an auto encoder (Fig. 2.1(a)). In this set up, photon was chosen to be the fundamental quanutm unit – in this case, qutrit. The three bases are formed from two distingushable optical paths and the lower-path photon's two polarizations respectively.

At the left hand side, there is a emitter (laser + BBO) which can produce single photon. This is essential as in order to obtain any relavent information about one photon's quantum state, each photon should be distinguished and measured individually; otherwise, it would not be possible to tell which properties belong to which photon and the desired statsitics would be unattinable [13]. The photons will first pass through some waveplate with predefined angle of incidence to pick up some

initial polarization angle. The photons then pass through beam a displacer and be put into two paths according to their currently polarizations: vertically polarized photons will go to the upper path due to a deflection angle while horizontally polarized photons will remain at the lower path because their optical path is not affected by the beam displacer. After another waveplate (orange colour) which create superposed state of vertically and horizontally polarized photons out of the horizontally polarized photons, the photons state vectors will have three bases:

$$V_{in} = \begin{bmatrix} v_1' \\ v_2' \\ v_3' \end{bmatrix}$$

where the coefficents of each basis depends on the incident angle of photon with respect to the three wave plates (green, blue and orange in Fig. 2.2(b)).

The autoencoder stage (Fig. 2.2(c)) is composed of another set of waveplates and beam dsiplacers. Basically these devices can change the photons' paths and polarizations and thereby change their state vectors. The waveplates are motorized, which means they can be adjusted to arbitrary angle with respect to the incoming photons and rendering photons in different quantum states.

The author devised a gradient descent algorithm to rotate the waveplates as to minimize the loss function, there the loss is defined with respect to the angles of four motorized waveplates at stage three (Fig. 2.2(c)). In other words, those angles are like the weights in a neual network. The loss function is defined as the average junk mode occupation probability over the different training states. Those training states can be viewed as a batch of input data. Since the task here is to compress the 3-level state vector (qutrit) into 2-level state vector (qubit), the objective is defined as minimizing the information at the third basis, which is defined as the 'junk mode' (Fig. 2.1(a), Fig. 2.2(c)). In this setup this mode is the uppermost path at stage three (Fig. 2.2(c)). In that state the photon is guaranteed to be vertically polarized. The quantum physical process guarantees that the entire transofrmation to the quantum states to be equivalent to some unitary transformation under ideal conditions (Fig. 2.1(b)).

2.3 Quantum Neural Networks

The underlining working principles of D. Bondarenko and P. Feldmann's quantum autoencoder [2] followed the formulation of the quantum neural networks (QNN) introduced by Beer et al. [4], which is fundamentally different from the classical autoencoder and the scheme proposed by this thesis in Section 3.1.

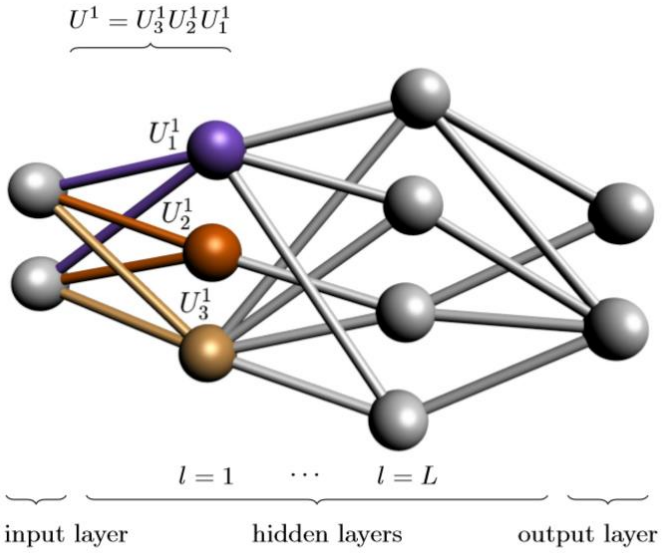

Fig. 2.3 Quantum Neural Networks Architecture (cited from [4])

The first difference is the input/output data/state vector representation. For a classical autoencoder, each neuron only contains a single numeric value; in our proposed design in Section 3.1, the entire layer is one state vector. In contrast, for QNN each neuron at input layer and output layer represents an entire qubit. This means that the size of the corresponding state vector will be exponentially larger. To be more specific, if the input layer has m neurons, the input state vector has a size of 2^m and same goes for output state vector size.

The second difference is the operation representation. For our design in Section 3.1, the unitary operation is represented by the feed forward flow pipeline (refer to Section 3.1). For QNN, each single neuron in hidden layer is a unitary matrix, denoted by U_i^l, where the superscript l indicates which layer the neuron is at and the subscript

i indicates the position of the neuron in top-down order (refer to Fig. 2.3). Furthermore, suppose the hidden layer has k neurons and the output layer has n neurons, the unitary equivalent of the entire QNN has the size of 2^{m+k+n} by 2^{m+k+n}. It is because the unitary matrix operates on the input state vector and the output state vector at the same time (the size contributed by the hidden layer neuron is for consistence of mathematical operations only and it does not affect the result). One might be misled by this design and think this architecture is not feed-forward. However, this QNN is indeed feedforward if the initial output state is set as $|00 \dots 00\rangle$ (default state) and take a partial trace on the entire system except the output after the evolution:

$$\rho^{out} \equiv tr_{in,hid}(U(\rho^{in} \otimes |00 \dots 00\rangle_{hid,out}\langle 00 \dots 00|)U^{\dagger}$$

where the evolution is dictated by the unitary equivalent of the hidden layer. The unitary equivalent U is defined as:

$$U \equiv \mathrm{U}^{out}\mathrm{U}^{L}\mathrm{U}^{L-1} \dots \mathrm{U}^{1}$$

where U with superscript is the unitary equivalent of each hidden layer and they are defined as:

$$\mathrm{U}^{l} \equiv U_1^l U_2^l U_3^l \dots U_i^l$$

The order of matrix multiplication must be strictly followed since those unitary operations do not commute in general. On a side notes, the hidden layer state (denoted by subscript 'hid') is automatically $|00 \dots 00\rangle$ because the information is already encoded in U.

The real reason that QNN is feedforward is that the output could be computed iteratively from left to right. It means that we can think of there is an output layer right after the first hidden layer (so the first hidden layer is also the last hidden layer). That output layer can be obtained by the aforementioned procedure and then treat it as the input to the second hidden layer and imagine there is another output right after the

second hidden layer (so the first hidden layer is also the last hidden layer) and obtain the second output. This process can be done iteratively until the output of the last hidden layer is obtained and the result would be identical to computing the output of the last hidden layer at one shot. In fact, the 'auxiliary' output layer can be inserted at arbitrary position with arbitrary times of iteration to facilitate computation, so long as the left-to-right order is followed.

2.4 Quantum Denoising Autoencoder (QDAE)

2.4.1 Objective

QDAE is devised to remove the noise from a noisy quantum state and subsequently produce a corresponding desired noiseless quantum state.

2.4.2 Network Architecture

Quantum autoencoders to denoise quantum data is the work done by D. Bondarenko and P. Feldmann [2]. They introduced a quantum counterpart of the well-studied classical autoencoder and investigated its performance on denoising quantum data (GHZ state to be more specific). Despite the drastically different working principle, the apparent architecture of their quantum denoising autoencoder (QDAE) surprisingly resembles its classical counterpart (Fig. 2.4):

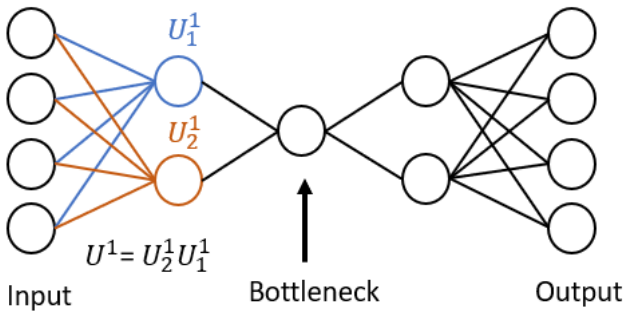

Fig. 2.4 Network Architecture of an QDAE (adapted from [2])

Ignoring that fact that each neuron is a unitary matrix, this architechture looks identical to a clasical autoencoder. As elaborated in Section 2.3, since the QDAE is just QNN with sepecific number of neurons at each layer, it is also inherently feed-forward. The process of the quantum information passes through each layer can be thought of as get compressed before the bottleneck and get reconstructed after the bottleneck. As such, to simpify the reasoning, QDAE can be thought of functioning in the same way as a classical denoising autoencoder (DAE) in the context of training the network.

2.4.3 Data Representation

In principle, since QDAE belongs to the so-called Quantum-Quantum Machine Learning (QQML) paradigm, the input data should be quantum data - namely qubits. However, in the two works related to DQN and QDAE, only simulation of the QQML was demonstrated since the complexity of the quantum device described here is beyond reach of the currently fabrication technology in near term. Therefore, the data representation is similar to that described in Section 3.1. The simulation of QDAE and its training is based on QETLAB [14] and the code is available on GitHub [15]. We modified the code in the repository for further investigation, as will be shown in Section 3.2. In the original work, the input states were chosen to be GHZ states.

2.4.4 Noise Type

The type of noise introduced in the work is called x-flip noise. X-flip noise is a common type of flipping noise in quantum channel. It can be represented by the following unitary matrix:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

X-flip noise will change state $|0\rangle$ to $|1\rangle$ and state $|1\rangle$ to $|0\rangle$ by the following operations:

$$\sigma_x|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$\sigma_x|1\rangle = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \begin{bmatrix}1\\ 0\end{bmatrix} = |0\rangle$$

It can also act on more than one qubit according to the following way:

$$(\sigma_x \otimes \sigma_x \otimes \dots \otimes \sigma_x)(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle)$$
$$= (\sigma_x|\psi_0\rangle) \otimes (\sigma_x|\psi_1\rangle) \otimes \dots \otimes (\sigma_x|\psi_n\rangle)$$

For example,

$$(\sigma_x \otimes \sigma_x \otimes \dots \otimes \sigma_x)(|0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle) = (\sigma_x|0\rangle) \otimes (\sigma_x|0\rangle) \otimes \dots \otimes (\sigma_x|0\rangle)$$
$$= |1\rangle \otimes |1\rangle \otimes \dots \otimes |1\rangle = |11 \dots 1\rangle$$

In the original work, only one x-flip was applied for one input GHZ state. It can be expressed as:

$$(I^{\otimes \mathrm{i}} \otimes \sigma_x \otimes \mathrm{I}^{\otimes \mathrm{j}})|\mathrm{GHZ}\rangle_M$$

where I denotes 2-by-2 identity matrix, $|\mathrm{GHZ}\rangle_M$ denotes a GHZ state composed by M number of qubits and with dimensionality of $2M$ (each qubit is two-dimensional), i and j are integers and $i + j + 1 = M$.

Furthermore, the action of x-flip is non-deterministic, meaning that there is only certain probability for which x-flip has effect. Therefore, the expected state after applying x-flip is:

$$|\mathrm{GHZ}\rangle_{ave} = p(I^{\otimes \mathrm{i}} \otimes \sigma_x \otimes I^{\otimes \mathrm{j}})|\mathrm{GHZ}\rangle_M + (1-p)|\mathrm{GHZ}\rangle_M$$

where p is the probability of x-flip having effect. If further assume different x-flipped corrupted states all have the same probability of occurrence, then the probability of a certain x-flipped corrupted state occurring is $\frac{p}{M}$.

2.4.5 Fidelity

The fidelity of noisy state is defined is the absolute square of its inner product with the noiseless state of the same kind:

$$F = |\langle\psi||\phi\rangle|^2 = \langle\phi|\rho|\phi\rangle$$

where ψ is the noisy state, ϕ is the desired noiseless state and ρ is the density matrix of ψ. The inner product follows Bra-ket (Dirac) notation. The maximum value of F is 1 (when two vectors are identical, their inner product is 1) and the minimum value is 0 due to the absolute square.

2.4.6 Cost Function

The cost function of the training algorithm is defined based on the fidelity of the output state from QDAE since it helps to have a metric to measure how good is the denoising performance:

$$C = \frac{1}{N}\sum_{x=1}^{N}\langle\phi_x|\rho_x{}^{out}|\phi_x\rangle$$

where x is the index of the input state and N is the batch size. In this case, the task is to maximize C. In this thesis, 'cost function' and 'loss function' are defined with the same definition: the objective function to be minimized.

2.4.7 Training algorithm

The Training algorithm of QDAE follows that of QNN, which can be found in Appendix D of literature [4].

2.4.8 Caveat

It needs to be noticed that, unlike in some of the classical DAE cases where the autoencoder can even reconstruct inputs it has never seen before, in the original work, the implementation of QDAE can only construct inputs of the same state as it has seen during the training stage with the same types of noises and the input states for training are unique (all are GHZ states). However, it does not require noiseless state vector as target value during the training process. Instead, part of the noisy states in procession are used as target values. The reason for doing so will be mentioned in Section 2.4.9. Therefore, $|\phi_x\rangle$ is no longer noiseless but is picked from the existing dataset. However, with this setup QDAE can still learn the representation of the noiseless state. The intuition is as follows:

Suppose the noiseless states are used as target value. Since there is only one type of target value (GHZ state), the final result of training would be just all the inputs being overfitted to put GHZ state (i.e. the training accuracy will be 100%). Needless to say, this is a trivial task (in classical situation but not in quantum case as will be explained in Section 2.4.9) since the QDAE would ignore the inputs during testing stage and simply output the same target value.

Supposed now GHZ states corrupted by x-flip in some arbitrary way (in $(I^{\otimes \mathrm{i}} \otimes \sigma_x \otimes \mathrm{I}^{\otimes \mathrm{j}})|\mathrm{GHZ}\rangle_M$ i and j are chosen arbitrarily) are used as the target values. Since the corrupted qubit can appear anywhere in a totally random manner, there is no way for the QDAE to overfit the target values (i.e. the training accuracy will not be 100%). However, QDAE will try to maximize C by maximizing its chance to hit the target value. When p is small, the best change to hit the target value is to become noiseless GHZ state since its occurrence is the highest compared to the occurrences of the rest states (i.e. different x-flipped corrupted states. Here assume that different x-flipped corrupted states all have the same chance of occurring). Therefore, the optimal output of QDAE after training is still the noiseless GHZ state. This point will be discussed quantitatively in Section 3.2.

2.4.9 Motivation of QDAE

One good question might be raised against this approach would be: why use a denoising autoencoder that cannot generalize to other type of input data in the first place? There is one naïve way for find the noiseless state of interest with even higher efficiency - that is to use majority vote. This principle has been widely adopted in digital communication field and known as cyclic redundancy check (CRC). The basic idea of this method is to send duplicate data for the same information and choose the majority which have the same information at the receiver end determine which what is the original information being transmitted. To demonstrate, suppose we would like to transmit a '1' bit, then 10 bits of 1 is transmitted with some extra header bit to indicate these five bits are in fact for the same information. Assume that during the transmission, noises are introduced. The probability that the transmitted bits being corrupted by bit is in general low, so it can be safely assumed that less than half of the total bits – say three bits for sake of the argument – was corrupted. Therefore, at the receiver end, the result is seven-bit '1' and three-bit '0'. Now by counting and doing a majority vote, it can be confidently concluded that '1' is the original information being sent out.

Similar to the classical scenario, when a quantum state being represented as a state vector was corrupted with relatively low noise (can be either flipping noise, phase noise or both), the state vector representation changed. Suppose we send out one hundred identical state vectors x and the total probability of them being corrupted is 0.3, there will be around seventy x remain at the receiver side. By counting and doing a majority vote, it seems that it would be a trivial task to tell what the original state is (x).

To answer that question, the context of the problem first needs to be understood. In quantum mechanics, measuring the state of a quantum object is not a trivial task. In order to do so, there should be a vast number of identical such state at disposal and partitioned into three group. After which, three orthogonal measurements (along x, y, z-direction) are to be carried out respectively to those states/qubits, each measurement

to one group. The so-called state is in fact a statistical result which can be concluded from all of these measurements. According to no-cloning theorem, there is no way to prepare identical copies for each of the received state [16]. Furthermore, after being measured, the quantum state will undergo a so-called ‘decoherence’ process and be destroyed (lose its quantum property). Therefore, such classical method would never work in this case. Moreover, in quantum engineering, preparation of quantum data is not a trivial task as well. It requires a lot of technical effort to engineer one single qubit of desired state. Therefore, our objective is not to recognize which state is the original, uncorrupted state; rather, the end is to have more such states. Therefore, denoising quantum autoencoder is used first to learn the representation of noiseless desired state, and then create more such states from the noisy, corrupted states given.

The conventional denoising autoencoder is technically not an unsupervised machine learning technique, since it requires ideal input data as the target value. In comparison, this proposed quantum denoising autoencoder works in a fully unsupervised manner. Imagine how expensive it would be if a noiseless quantum state is used as target value for training (assume it is possible to prepare one in the first place, which is practically impossible due to aforementioned reasons), as for each input training data, one perfect desired state needs to be measured and destroyed. Price is way larger than the gain and it wouldn’t worth trying to do it in the first place. On the other hand, measuring the fidelity of a quantum state can be done in a one-shot manner, so it is possible to the proposed scheme to work [17].

Chapter 3

Improved/Alternative Approaches to the Existing Works

3.1 Alternative Approach for Compressive Quantum Autoencoder

We propose an alternative approach to train a quantum autoencoder. While the original work [1] is within QQML paradigm, it is possible to reformulate the task into a QCML problem, namely, to use classical machine learning techniques to be the horsepower.

3.1.1 Tools

Tool learned and used: Theano

As a Python library and an optimizing compiler, Theano is commonly used for evaluating and manipulating mathematical expressions, especially matrix-valued ones [18]. In Theano, computations are expressed using a NumPy-esque syntax and compiled to run efficiently on either CPU or GPU architectures.

3.1.2 Constrained Optimization Pipeline Formulation

The physical problem has been formulated in terms of a constrained optimization problem. First, the problem is defined as such:

Given a set of vectors which do not span the entire Hilbert space, find a unitary transformation that can use fewer number of basis to describe these vectors. 'Do not span the entire Hilbert space' simply means that these vectors are in principle compressible in a lossless manner. The real reason a unitary transformation is required is that all manipulations on quantum states must be equivalent to some form of unitary operation to ensure the reversibility and to preserve the vector norm. On the other hand, unitary transformation can be interpreted in a geometrical sense as some rotation on vectors. Hence it is not difficult to see if the given set of vectors does not space the entire space in which all possible vectors exist, it is always possible to find a rotation to re-align their bases to the canonical bases and it only takes a few the canonical bases to describe them, ergo, lossless compression. However, if the problem is to be solved in constrained optimization paradigm, closed-form solution is not available and the final result is an approximation of the real lossless compression. How close it is to the optimal result depends on how good our strategy is.

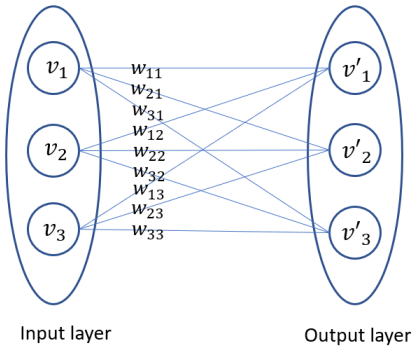

Fig. 3.1 Neural network for constrained optimization

We define a two-layer perceptron as shown in Fig. 3.1, each layer with three nodes for simplicity. The first layer is the input layer and the three nodes have the values of each

entry of the initial 3-dimential input vector. The second layer is the output layer and the three nodes have the values of each entry of the final output vector. Since the input-output layers are fully connected, there are 9 connections and 9 weights in total. Our objective is to compress the input vector. One way is to set one constraint as one of the output nodes to have a minimum value (ideally 0). The second constraint is that the output vector has the same norm as the input vector as unitary transformation preserves the norm. In the case of state vector for a quantum system, the norm is always 1. Now the output vector V_{out} can be represented by the following relation with the input vector V_{in}:

$$V_{out} = \begin{bmatrix} v'_1 \\ v'_2 \\ v'_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = WV_{in}$$

Notice that all the weights collectively can be thought as entries of some square matrix W that operates on V_{in}. W is called the weight matrix. This coincides with our objective of finding some specific transformation, which can be viewed as a matrix under certain assigned bases. In fact, it can be proved that if V_{in} and V_{out} have the same norm, W is indeed unitary. Further notice that in the context of deep learning, the activation function is always one in order to have the relation in equation (1).

Now it is rather straight forward to specify the objective function to be optimized. It would be to minimize such a squared-error loss function:

$$L = \sum ([w_{11} \quad w_{12} \quad w_{13}]V_{in})^2 + \lambda(([w_{21} \quad w_{22} \quad w_{23}]V_{in})^2 + ([w_{31} \quad w_{32} \quad w_{33}]V_{in})^2 - 1)^2$$

The first term is to minimize the first entry of V_{out} (ideally 0). The second term is to impose the unit norm constraint. λ can be considered as the Lagrange multiplier which

tunes the relative importance of the second term with respect to the first term (i.e. Which constraint has the priority in the optimization scheme).

Now perform gradient descent technique on the loss function, differentiating it with respect to all the weights in the hope of finding a local minimum.

3.1.3 Caveat

The gradient descent could and has been implemented in 2 following 2 forms:

1) Gradient descent in real space

This is the normal type of gradient descent which can be seen in many places.

2) Gradient descent in complex space

Since Theano itself does not support complex space gradient descent, the problem needs to be formulated in a different way:

Define a complex number $z = x + iy$. Then

$$dz = dx + idy$$

and

$$\frac{\partial}{\partial \mathrm{z}} = \frac{\partial}{\partial \mathrm{x}} + i\frac{\partial}{\partial \mathrm{y}}$$

Since differentiation is a linear operation. It means that the real part and the complex part of the weights can be treated separately.

However, things get more complicated for complex number multiplication. Assume $z_1 = x_1 + iy_1$ and $z_2 = x_2 + iy_2$, then

$$\begin{aligned} z_1 z_2 &= (x_1 + iy_1)(x_2 + iy_2) \\ &= x_1 x_2 + iy_1 x_2 + ix_1 y_2 - y_1 y_2 \\ &= (x_1 x_2 - y_1 y_2) + i(x_2 y_1 + x_1 y_2) \end{aligned}$$

and

$$\begin{aligned} z_1 z_2{}^* &= (x_1 + iy_1)(x_2 - iy_2) \\ &= x_1x_2 + iy_1x_2 - ix_1y_2 + y_1y_2 \\ &= (x_1x_2 + y_1y_2) + i(x_2y_1 - x_1y_2) \end{aligned}$$

where $z_2{}^*$ is the complex conjugate of z_2.

Therefore, by abiding to this rule, it is possible to define to variables x and y to going through the gradient descent process and combine the final result to give the adjusted complex number z.

3.1.4 Real-space Gradient Descent

The input data are 3-dimensional unit vectors with all real-valued entries. Moreover, it satisfies the constraint that they all live in the same 2-dimensional plane, which may or may or coincide with the plane spanned by the basis vectors (i.e. x-y, y-z and x-z plane). It this achieved by generating random unit vectors on x-y plane first with 3-dimensional representation (e.g. $[\sqrt{0.5}, \sqrt{0.5}, 0]$]), then apply 3-dimensional rotational matrices:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{bmatrix}$$
$$R_y = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 \\ 0 & 1 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix}$$
$$R_z = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

sequentially with arbitrarily selected real-valued θ_1, θ_2, θ_3.

The numerical optimization was executed for 1000 iterations, each with 1 input vector. The matrix obtained was tested on another 100 vectors generated by the same method (same θ_1, θ_2, θ_3 for R_x, R_y, R_z applied on random unit vectors on x-y

plane). For the sake of argument, set $\theta_1 = 0.1$, $\theta_2 = 0.2$, $\theta_3 = 0.3$, Lagrange multiplier $\lambda = 9$ (arbitrarily), learning rate=0.01(empirically chosen).

The matrix obtained:

$$W = \begin{bmatrix} 0.0380191 & -0.00639751 & 0.169849 \\ 0.0616013 & 0.989382 & 0.298982 \\ 1.08257 & -0.0128492 & 0.25824 \end{bmatrix}$$

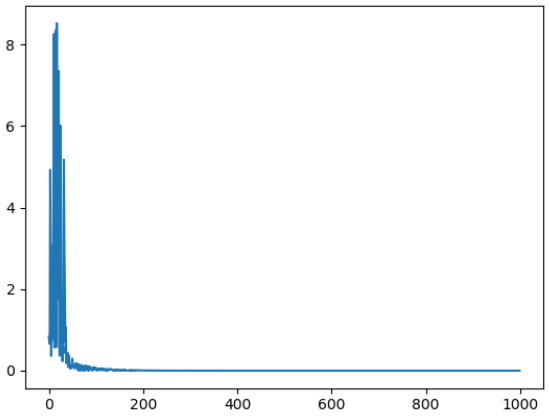

Fig. 3.2(a) Loss versus Iteration

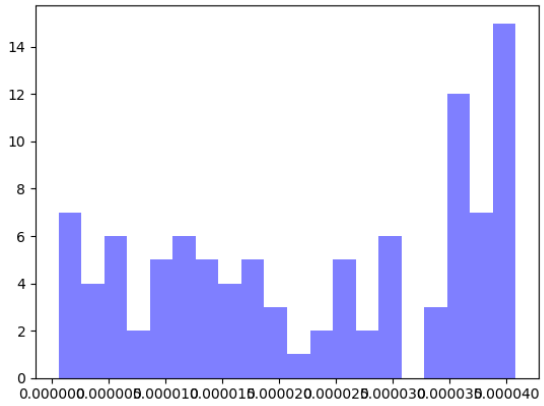

Fig. 3.2(b) Histogram of the absolute values of the first entry of the output vector during testing phase

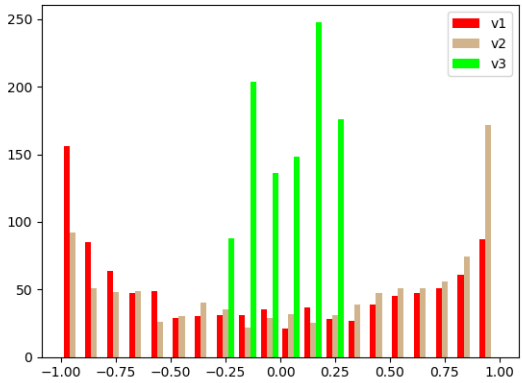

Fig. 3.3(a) Histogram of the of each entry of the 1000 input for training

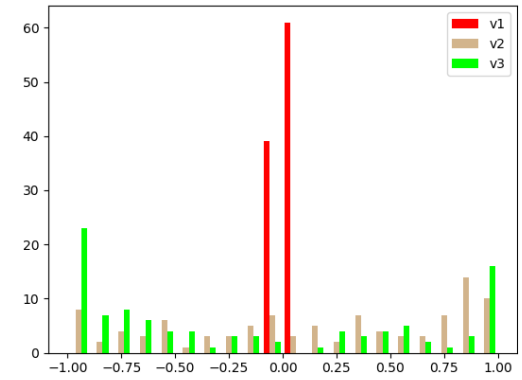

Fig 3.3(b) Histogram of the values of each entry of the 100 output vectors during testing phase

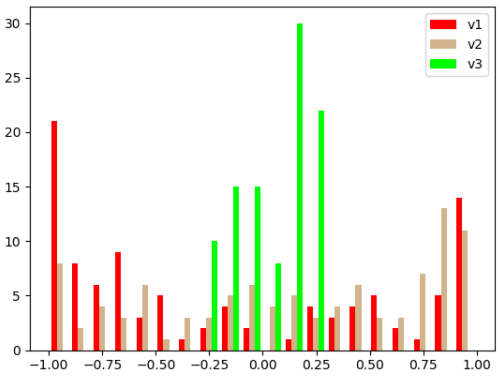

Fig 3.3(c) Histogram of the values of each entry of the 100 input vectors during testing phase

Norm variance: 3.685904882157093e-19

Norm mean: 1.0000000007092062

The results look promising. The mean of the norm of the 100 outputs for the 100 test vectors is around 1 and the variance is 0 for all intents and purposes.

Now check the unitarity of W:

$$W * W^{\dagger} = \begin{bmatrix} 0.0303349 & 0.04679401 & 0.08510218 \\ 0.04679401 & 1.07206218 & 0.13118386 \\ 0.08510218 & 0.13118386 & 1.23881049 \end{bmatrix}$$

$$W^{\dagger} * W = \begin{bmatrix} 1.17719783 & 0.04679387 & 0.30443766 \\ 0.04679387 & 0.97908353 & 0.29140229 \\ 0.30443766 & 0.29140229 & 0.18492622 \end{bmatrix}$$

Evidently, $W * W^{\dagger} \neq W^{\dagger} * W \neq I$. So W is not unitary. This demystify baffling result, the input test vectors (heads represented by red dots), output test vectors (heads represented by red dots), the basis vectors before transformation and the basis vectors after transformation are plotted:

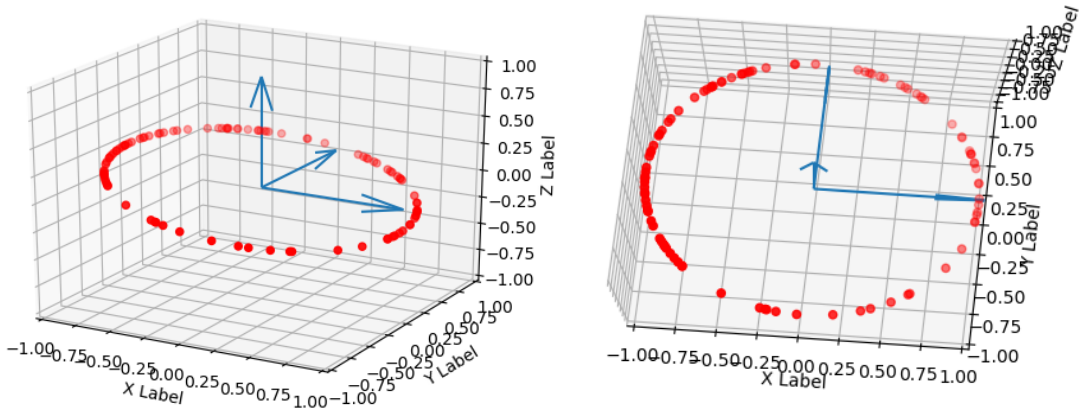

Fig. 3.4(a)(b) 3-D scattered plots of input data with basis vectors drawn view 1, 2

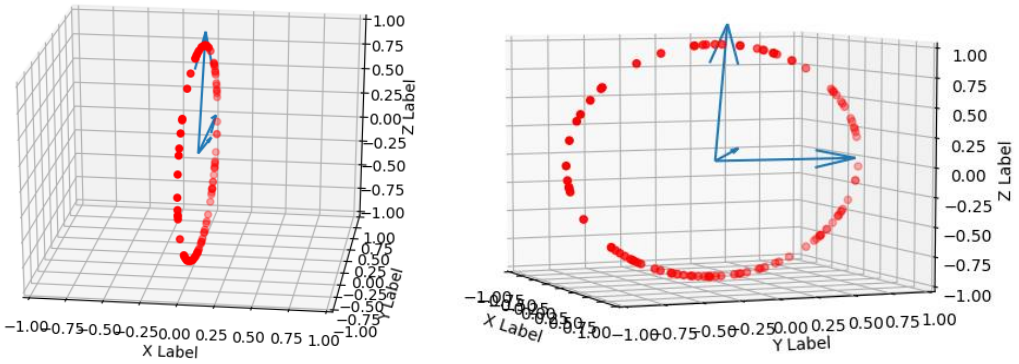

Fig. 3.4(c)(d) 3-D scattered plots of output data with the transformed basis vectors drawn view 1, 2

It can be seen that the ring of input vectors really gets ‘rotated’ to the y-z plane without distortion (same relative positioning). By taking the inner product of the two basis vectors near same plane as the ring, it can be shown that they are no longer orthogonal (inner product is 0.04679387110319424 rather than 0). Their respective norms are 1.1771978316293592 and 0.9790835259577558, which are no longer 1. As for the third basis vector, the distortion is even more severe. Its norm is 0.18492621547389604 and neither its angle with the other two basis vectors are near to 90 degree (more obvious to see with the 3d-scatter plot which supports observing from different angles, but unfortunately not able to show in the report). Its inner product for them are 0.30443765716909904 and 0.29140228568326726 respectively.

Based on those results, it can concluded that only the norm of the vectors in the same plane as the training/testing vectors and their relative positions (angles) are preserved.

Now it is clear that the constraints defined in the loss function are insufficient to impose the unitarity of the matrix. Now modify the loss function, such that the columns of W are normalized and they are orthogonal to each other (the inner products between any of the columns in 0):

$$L = \sum([w_{11} \quad w_{12} \quad w_{13}]V_{in})^2 + \lambda_1((w_1 * w_2)^2 + (w_1 * w_3)^2 + (w_2 * w_3)^2) + \lambda_2(\left(||w_1||^2 - 1\right)^2 + \left(||w_2||^2 - 1\right)^2 + \left(||w_3||^2 - 1\right)^2)$$

where w_1, w_2, w_3 are the columns of W. In this way the basis vectors' norms and the mutual orthogonality between any of the basis vectors are preserved.

The loss function now looks rather lengthy because of the additional constraints imposed. However, one of the original constraints

$\lambda_0(([w_{21} \quad w_{22} \quad w_{23}]V_{in})^2 + ([w_{31} \quad w_{32} \quad w_{33}]V_{in})^2 - 1)^2$

is no longer necessary. It is because the fact that the three basis vectors' norms are preserved after the transformation guarantees the any output vector in the space spanned by those basis vectors having their norms preserved. And since by the first constraint the first entry of the output vector is 0, the norm of the 2-d representation for output vector is definitely 1.

With the new loss function and the same hyperparameter setting (except 2000 iterations with totally 2000 training input vectors and all Lagrange multipliers set to 1), the following result is obtained:

$$W = \begin{bmatrix} -0.218418 & 0.03728117 & -0.97513878 \\ 0.86303829 & 0.47379897 & -0.17516862 \\ -0.45547247 & 0.87984154 & 0.13574648 \end{bmatrix}$$

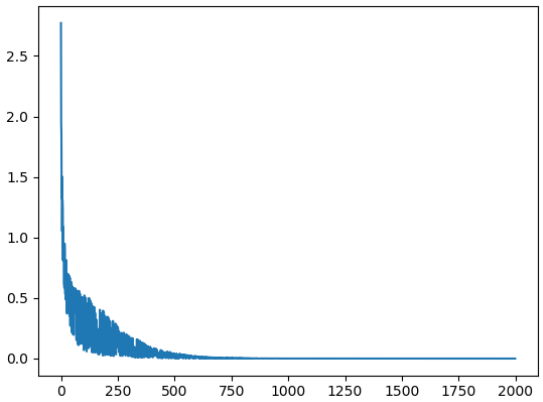

Fig. 3.5(a) Loss versus Iteration

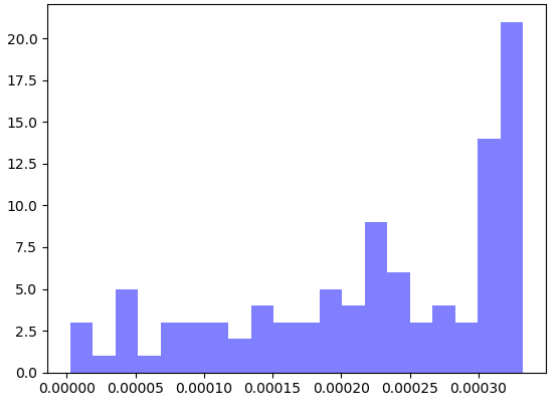

Fig. 3.5(b) Histogram of the absolute values of the first entry of the output vector during testing phase

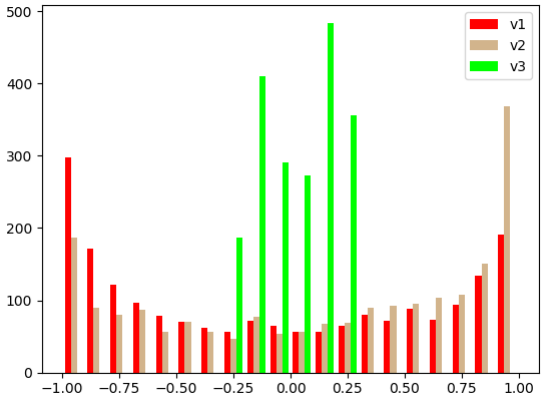

Fig. 3.6(a) Histogram of the of each entry of the 1000 input for training

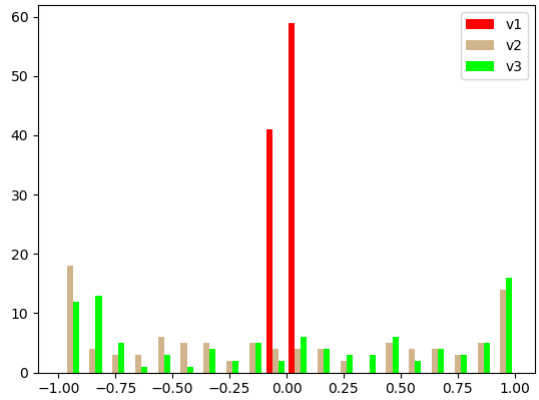

Fig. 3.6(b) Histogram of the values of each entry of the 100 output vectors during testing phase

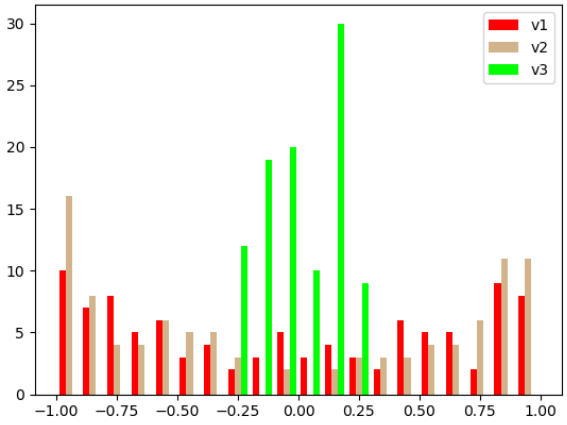

Fig. 3.6(c) Histogram of the values of each entry of the 100 input vectors during testing phase

Norm variance: 3.8916991203251375e-13

Norm mean: 1.0000038998648864

Now check the unitarity of W:

$$W * W^{\dagger} = \begin{bmatrix} 9.99991944e-01 & -2.56045504e-05 & -8.67404541e-05 \\ -2.56045504e-05 & 1.00000460e+00 & -6.88396491e-07 \\ -8.67404541e-05 & -6.88396491e-07 & 1.00000341e+00 \end{bmatrix}$$

$$W^{\dagger} * W = \begin{bmatrix} 9.99996685e-01 & 2.01737318e-05 & -1.81448650e-05 \\ 2.01737318e-05 & 9.99996488e-01 & 8.63593756e-05 \\ -1.81448650e-05 & 8.63593756e-05 & 1.00000679e+00 \end{bmatrix}$$

This time $W * W^{\dagger}$and $W^{\dagger} * W$ are almost I.

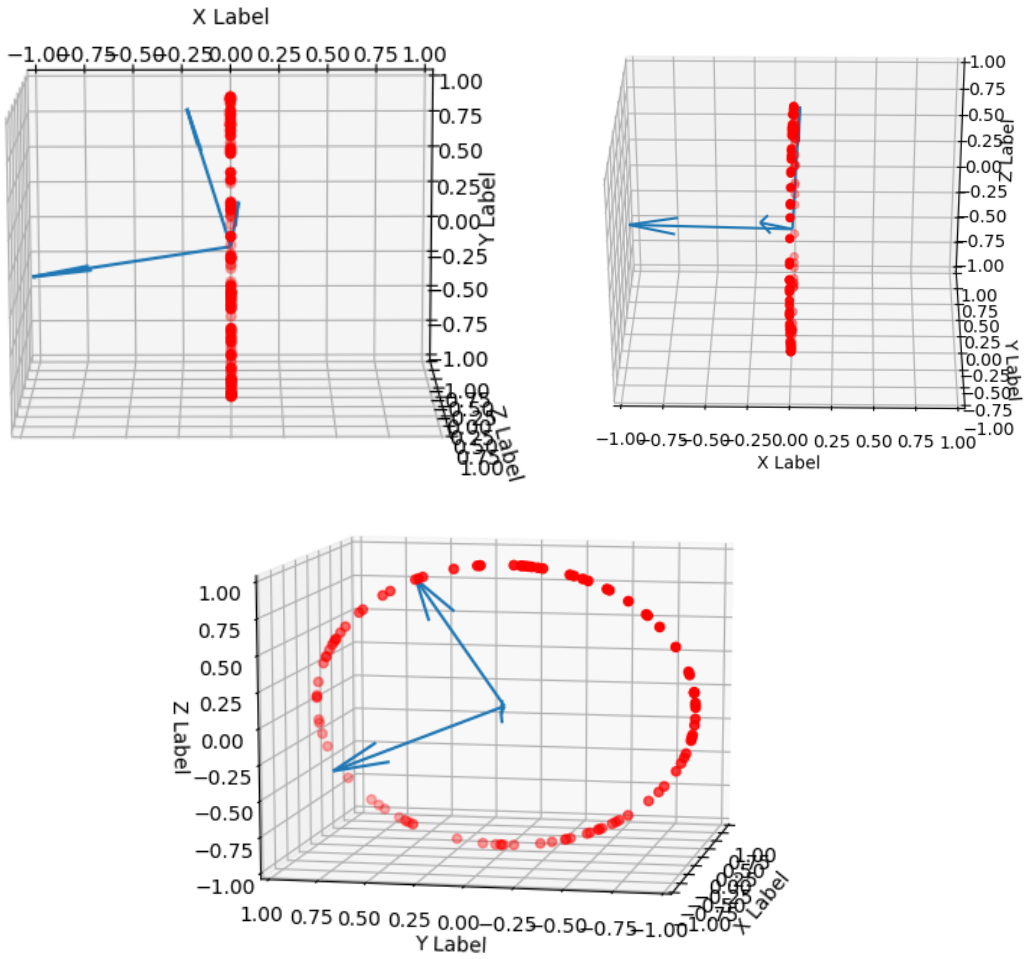

Fig. 3.7(a)(b)(c) 3-D scattered plots of output data with the transformed basis vectors drawn view 1, 2, 3

By observing the 3-d plot, it also can be seen that now the transformed basis vectors are perpendicular to each other. It is consequential as the unitary constraint is already satisfied (practically).

Therefore, for all intents and purposes, this weight matrix is qualified as a transformation to compress the given data.

3.1.5 Complex-space Gradient Descent

For complex-space gradient descent, 2-dimensional complex vector inputs are used as a toy example. This is due to the fact that implementing complex gradient descent is much complicated to define the loss function and the 2-dimensional case is more help to break down different segments and hence gain some insights. The objective is to transform a vector initially lies in R^2 space into C^1 space. This is supposed to be a rotation that compress the 2-d vector without loss of information as long as the exact transformation that did the job is known.

The loss function needs to be modified as well:

$$
\begin{aligned}
L = \sum \lambda_1 (& \left([w_{11\text{real}} \quad w_{12\text{real}}] V_{in(real)} - [w_{11\text{img}} \quad w_{12\text{img}}] V_{in(img)} \right)^2 \\
& + \left([w_{11\text{real}} \quad w_{12\text{real}}] V_{in(img)} + [w_{11\text{img}} \quad w_{12\text{img}}] V_{in(real)} \right)^2) \\
& + \lambda_2 ((w_{11\text{real}} * w_{12\text{real}} + w_{11\text{img}} * w_{12\text{img}})^2 \\
& + (w_{21\text{real}} * w_{22\text{real}} + w_{21\text{img}} * w_{22\text{img}})^2 \\
& + (w_{11\text{real}} * w_{12\text{img}} - w_{11\text{img}} * w_{12\text{real}})^2 \\
& + (w_{21\text{real}} * w_{22\text{img}} - w_{21\text{img}} * w_{22\text{real}})^2) \\
& + \lambda_3 (\left({w_{11\text{real}}}^2 + {w_{11\text{img}}}^2 + {w_{21\text{real}}}^2 + {w_{21\text{img}}}^2 - 1 \right)^2 \\
& + \left({w_{12\text{real}}}^2 + {w_{12\text{img}}}^2 + {w_{22\text{real}}}^2 + {w_{22\text{img}}}^2 - 1 \right)^2)
\end{aligned}
$$

where w_{ijreal} and w_{ijimg} represents the real and imaginary part of the weight matrix elements. This loss function adapts the complex number multiplication rules, to enforce the same constraint as before (minimize the input vectors' first entries, normalization and mutual orthogonality of each column of the weight matrix. Then treat each all the variables as if they were just ordinary real variables and carry out the gradient descent. The results are shown below:

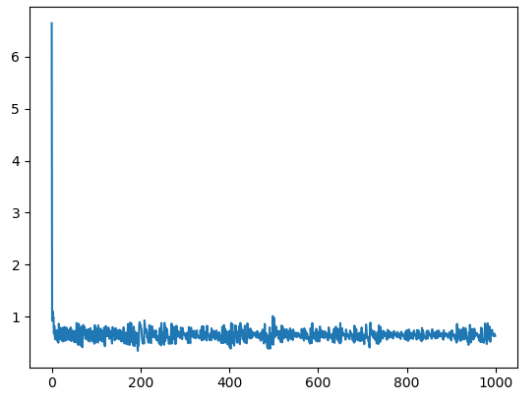

Fig. 3.8(a) Loss versus Iteration

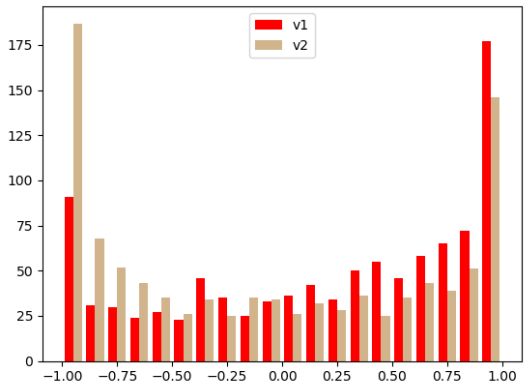

Fig. 3.8(b) Histogram of the absolute value of each entry of the 1000 input vectors for training

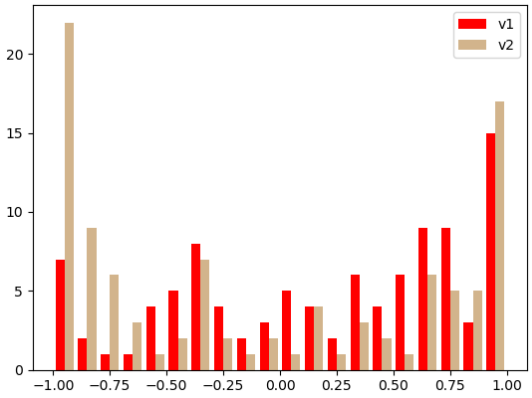

Fig. 3.8(c) Histogram of the absolute value of each entry of the 100 input vectors during testing phase

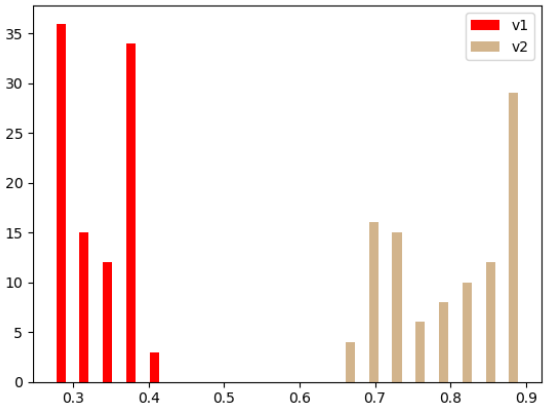

Fig. 3.8(d) Histogram of the absolute value of each entry of the 100 output vectors during testing phase

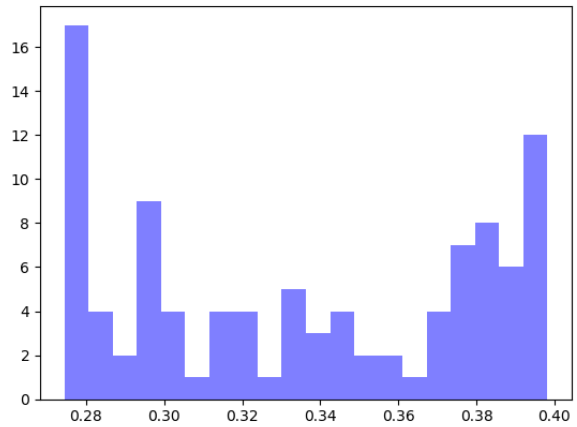

Fig. 3.8(e) Histogram of the absolute values of first entries of the output vector during testing phase

Norm variance: 0.09657760930611538

Norm mean: 0.7889423631808617

From the results, it can be seen that the gradient descent was not executed well. After immediately dropped by a lot, the loss kept fluctuating between 0.5 and 1 and there is no sign to deceasing further. Compared to the final loss for real-space gradient descent case, which is almost 0, this loss is not acceptable. This view is further strengthened by Fig.3.8(e). The first entry of the vector is far from 0. At the same time, neither norm mean nor norm variance is within an acceptable range. These are the direct evidences that the compression has failed.

In order to see why that is the case, the loss function is investigated. Some of the constraints are relaxed to dissect the loss function. To begin with, relax the constraint that normalize the columns of weight matrix by setting $\lambda_3 = 0$:

$$L = \sum \lambda_1 \left([w_{11\text{real}} \quad w_{12\text{real}}] V_{in(real)} - [w_{11\text{img}} \quad w_{12\text{img}}] V_{in(img)} \right)^2$$
$$+ \left([w_{11\text{real}} \quad w_{12\text{real}}] V_{in(img)} + [w_{11\text{img}} \quad w_{12\text{img}}] V_{in(real)} \right)^2$$
$$+ \lambda_2 ((w_{11\text{real}} * w_{12\text{real}} + w_{11\text{img}} * w_{12\text{img}})^2$$
$$+ (w_{21\text{real}} * w_{22\text{real}} + w_{21\text{img}} * w_{22\text{img}})^2$$
$$+ (w_{11\text{real}} * w_{12\text{img}} - w_{11\text{img}} * w_{12\text{real}})^2$$
$$+ (w_{21\text{real}} * w_{22\text{img}} - w_{21\text{img}} * w_{22\text{real}})^2)$$

and execute gradient descent. The results are shown below:

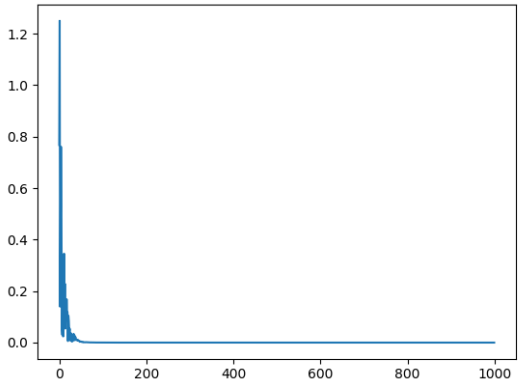

Fig. 3.9(a) Loss versus Iteration

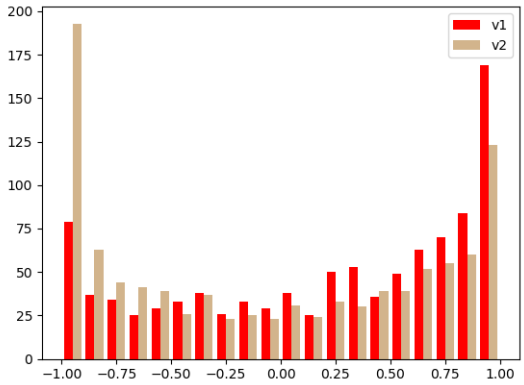

Fig. 3.9(b) Histogram of the absolute value of each entry of the 1000 input vectors for training

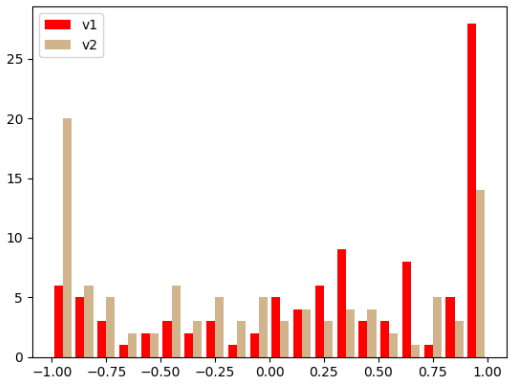

Fig. 3.9(c) Histogram of the absolute value of each entry of the 100 input vectors during testing phase

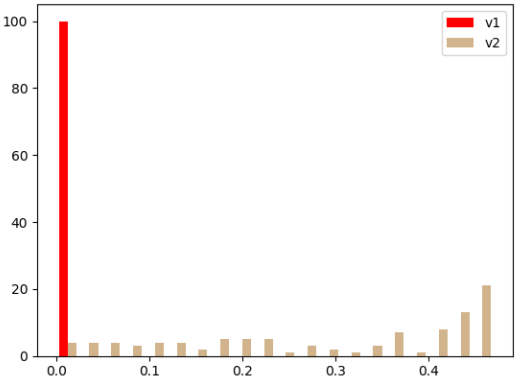

Fig. 3.9(d) Histogram of the absolute value of each entry of the 100 output vectors during testing phase

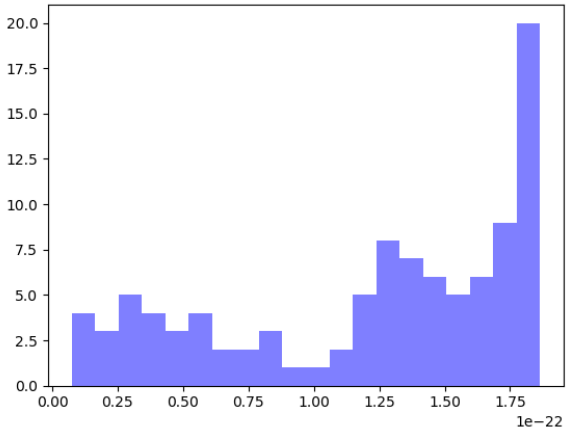

Fig. 3.9(e) Histogram of the absolute values of first entries of the output vector during testing phase

Norm variance: 0.006871380406262313

Norm mean: 0.11094007653732647

This time the loss deceased to near 0 eventually with little fluctuation. From Fig. 3.9(e) it can be seen that the norm for the first entry is also near 0. To check the inner product between the two columns of W:

```
In [97]: U[:,0].getH().dot(U[:,1])
Out[97]: matrix([[-4.42692236e-12-2.12832974e-11j]])
```

Fig. 3.10 Inner product of columns of W

Indeed, the two columns are orthogonal to each other for all intents and purposes. The results suggest that zero value constraint and orthogonality constraint can both be satisfied at the same time. The norm mean is nowhere near 1 and variance is not negligible, but this is expected since the normalization constraint is dropped at this point.

Now relax the mutual orthogonality constraint by setting $\lambda_2 = 0$:

$$L = \sum \lambda_1 (([w_{11\text{real}} \quad w_{12\text{real}}]V_{in(real)} - [w_{11\text{img}} \quad w_{12\text{img}}]V_{in(img)})^2 + ([w_{11\text{real}} \quad w_{12\text{real}}]V_{in(img)} + [w_{11\text{img}} \quad w_{12\text{img}}]V_{in(real)})^2) + + \lambda_3 ((w_{11\text{real}}{}^2 + w_{11\text{img}}{}^2 + w_{21\text{real}}{}^2 + w_{21\text{img}}{}^2 - 1)^2 + (w_{12\text{real}}{}^2 + w_{12\text{img}}{}^2 + w_{22\text{real}}{}^2 + w_{22\text{img}}{}^2 - 1)^2)$$

and execute gradient descent. The results are shown below:

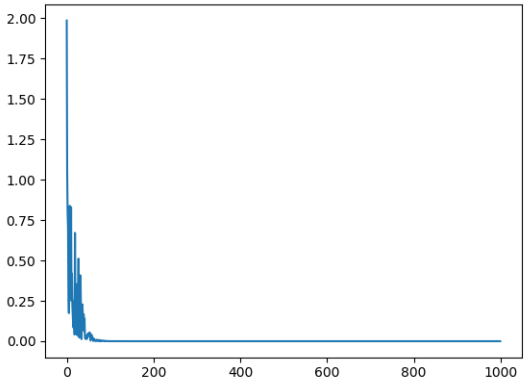

Fig. 3.11(a) Loss versus Iteration

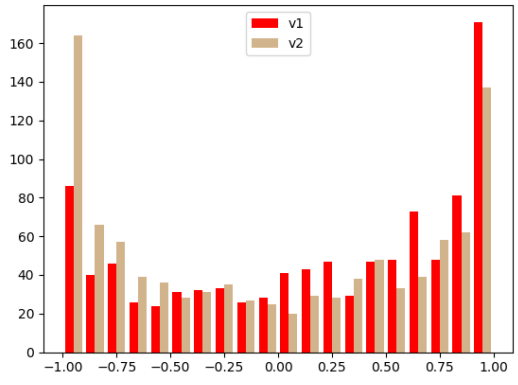

Fig. 3.11(b) Histogram of the absolute value of each entry of the 1000 input vectors for training

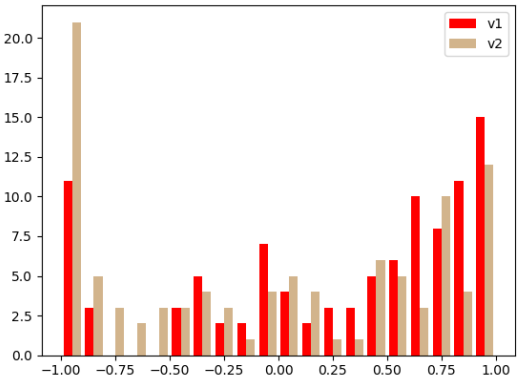

Fig. 3.11(c) Histogram of the absolute value of each entry of the 100 input vectors during testing phase

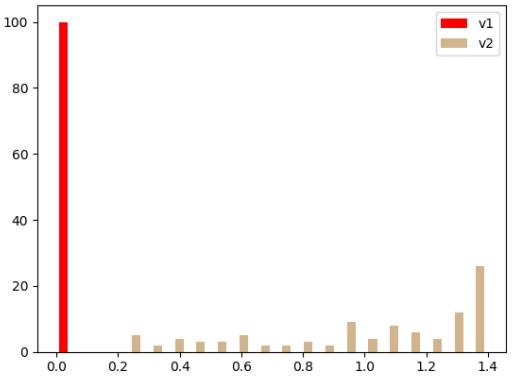

Fig. 3.11(d) Histogram of the absolute value of each entry of the 100 output vectors during testing phase

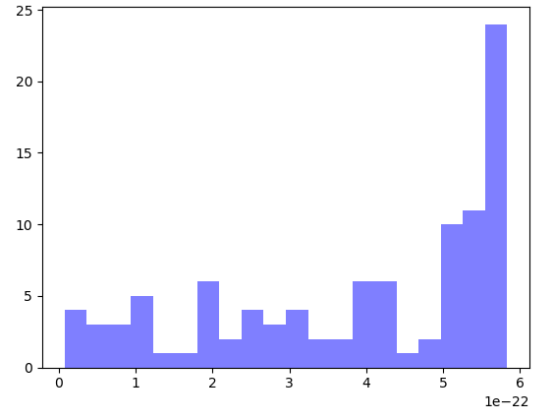

Fig. 3.11(e) Histogram of the absolute values of first entries of the output vector during testing phase

Norm variance: 0.42523339506248653

Norm mean: 1.1358641944233774

The loss behaves normally on the loss-versus-iteration plot. The absolute value of the first entry of the output vectors are also near to 0. The first constraint satisfied. To check if the normalization constrain is satisfied, to check the norms of both columns of W:

```
In [77]: format((abs(W[:,0]).T*abs(W[:,0]))[0,0], '.12g')
# give 12 significant digits
Out[77]: '1'
```

Fig. 3.12(a) Norm of the first column of columns of W

```
In [83]: format((abs(W[:,1]).T*abs(W[:,1]))[0,0], '.12g')
# give 12 significant digits
Out[83]: '1'
```

Fig. 3.12(b) Norm of the second column of columns of W

Indeed, the normalization constrain is satisfied. This means the first the third constraints can be satisfied simultaneously.

However, the norm mean and norm variance suggest that the output vectors' norms are not normalized. Since it is not easy to visualize C^2 space, R^3 space is used to illustrate the reason.

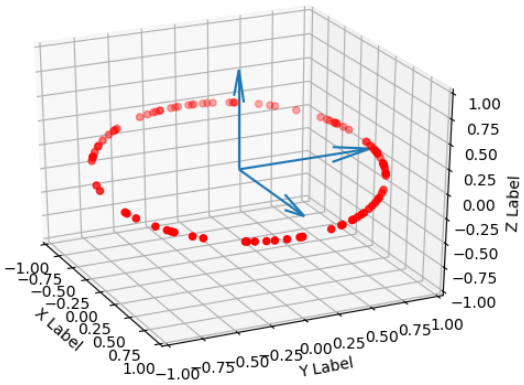

Fig. 3.13(a) 3-D scattered plots of input data with basis vectors drawn

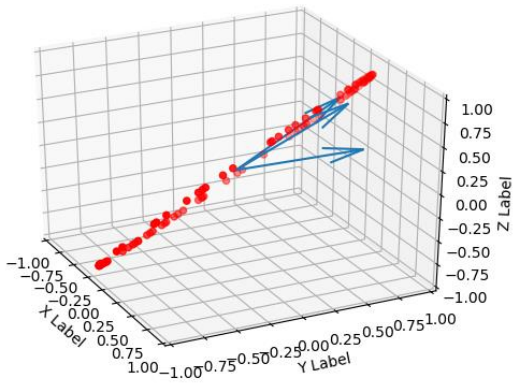

Fig. 3.13(b) 3-D scattered plots of output data with the transformed basis vectors drawn

Like in the real-space gradient descent case, the loss is to be minimized function. But now the loss function is defined as:

$$L = \sum([w_{11} \quad w_{12} \quad w_{13}]V_{in})^2 + \lambda_2((\left|\left|w_1\right|\right|^2 - 1)^2 + (\left|\left|w_2\right|\right|^2 - 1)^2 + (\left|\left|w_3\right|\right|^2 - 1)^2)$$

By setting λ_1 to 0. So similar to the complex-space case, it also gets rid of the orthogonality constraint. It can be seen that despite the normalization constraint is still present, the norms of the majority of the output vectors are not preserved despite the fact that the three bases vectors norms are still 1 (Fig. 3.13(a) and Fig. 3.13(b)). The factors are squeezed to a line to satisfy the compression constraint while the three basis vectors' relative positions changed (no longer orthogonal) but their norms are preserved. This example intuitively shows use that the normalization constraint alone is not enough to preserve the norm of the input vectors.

Finally, relax the first constraint which minimize the first entry of the input vectors by setting λ_1 to 0 for our loss function for complex case:

$$L = \sum \lambda_2 \left((w_{11\text{real}} * w_{12\text{real}} + w_{11\text{img}} * w_{12\text{img}})^2 \right.$$
$$+ (w_{21\text{real}} * w_{22\text{real}} + w_{21\text{img}} * w_{22\text{img}})^2$$
$$+ (w_{11\text{real}} * w_{12\text{img}} - w_{11\text{img}} * w_{12\text{real}})^2$$
$$\left. + (w_{21\text{real}} * w_{22\text{img}} - w_{21\text{img}} * w_{22\text{real}})^2 \right)$$
$$+ \lambda_3 \left(\left({w_{11\text{real}}}^2 + {w_{11\text{img}}}^2 + {w_{21\text{real}}}^2 + {w_{21\text{img}}}^2 - 1 \right)^2 \right.$$
$$\left. + \left({w_{12\text{real}}}^2 + {w_{12\text{img}}}^2 + {w_{22\text{real}}}^2 + {w_{22\text{img}}}^2 - 1 \right)^2 \right)$$

and carry out gradient descent. The results are shown below:

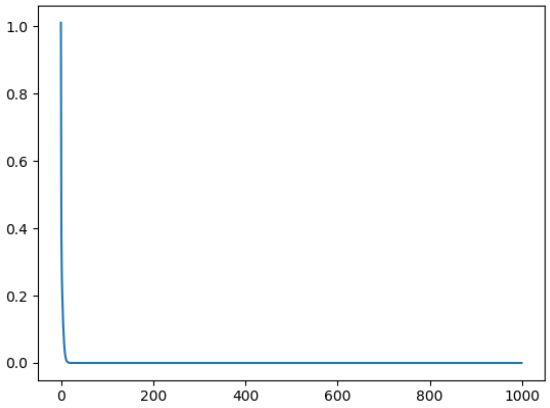

Fig. 3.14(a) Loss versus Iteration

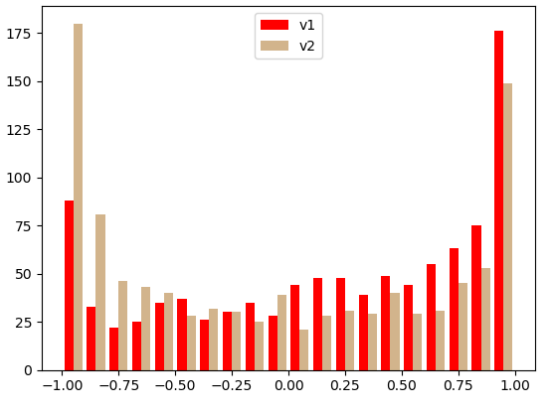

Fig. 3.14(b) Histogram of the absolute value of each entry of the 1000 input vectors for training

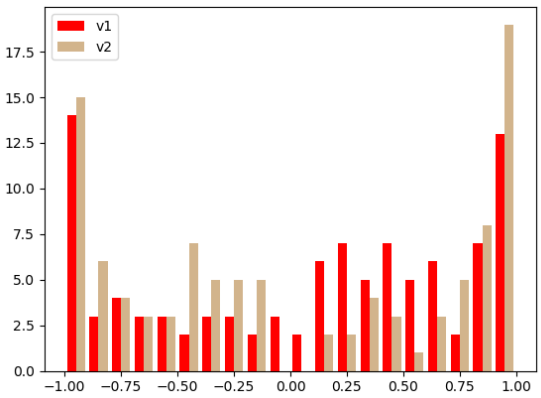

Fig. 3.14(c) Histogram of the absolute value of each entry of the 100 input vectors during testing phase

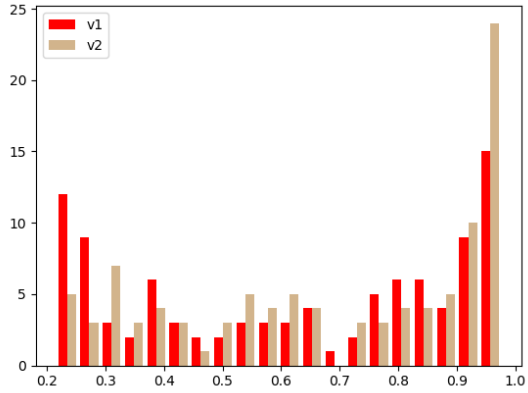

Fig. 3.14(d) Histogram of the absolute value of each entry of the 100 output vectors during testing phase

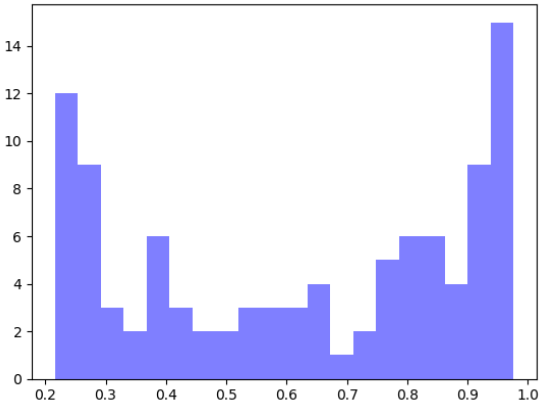

Fig. 3.14(e) Histogram of the absolute values of first entries of the output vector during testing phase

Norm variance: 4.831773044478697e-32

Norm mean:1.0

From the loss plot, the constraints are satisfied very quickly. It can be seen that the norm mean and variance shows the input vector's norms are preserved perfect. This is because this time W is unitary. To verify, compute

$$W * W^{\dagger} = \begin{bmatrix} 1.0000000e+00+4.190529e-18j & 8.673617e-17-5.551115e-17j \\ 8.6736173e-17+2.775557e-17j & 1.000000e+00+1.893625e-17j \end{bmatrix}$$

where

$$W = \begin{bmatrix} 0.9280115603+0.2752480966j & -0.1823128477+0.1726124408j \\ -0.0964604334+0.2317939040j & 0.5873887857+0.7693772710j \end{bmatrix}$$

Indeed, W is unitary, and it means that normalization and orthogonality constraints can be satisfied simultaneously for complex matrix as well. However, it is just come random unitary matrix which perform some random rotation in C^2 space, since there is no constraint to compress the input vector.

3.1.6 Observations

After detailed investigation, it becomes clear that:

1. Any 2 of the given 3 constraint can be satisfied simultaneously;
2. All 3 constraints cannot be satisfied simultaneously.

This observation forces us to conclude that the unitary matrix that can perform R^2-to-C^1 rotation does not exist. After a bit further research online, it was found out that R^{2n} spaces are not congruent to C^n space despite they share a lot of common properties. This implies that certain rotation in R^{2n} are not possible in C^n and the one needed for the designed compression falls into this category.

3.1.7 Comparison with PCA

Principal component analysis (PCA) is a very commonly practiced unsupervised data dimensionality reduction technique. Here an analogy is drawn between the technique used in this project and PCA.

For starters, the following question is to be asked: is it possible to solve our task with PCA. Essentially, what PCA does is to find a set of new basis vectors to form new coordinates to represent the data, such that the first k basis vectors will capture the most variance of the data distribution. It does so by finding the eigenvectors of the covariance matrix of the given data, and those basis vectors that capture the most variance is the eigenvectors of the covariance matrix which corresponds to the largest eigenvalues.

For our constraint data distribution, it is in fact pretty straightforward for PCA to find the new basis which can represent the data in a lower dimensional vector space.

For example, in the simplest non-trivial case, our data are distributed on a 2-dimensional ring centered at the origin and oriented arbitrarily in a 3-dimentional space:

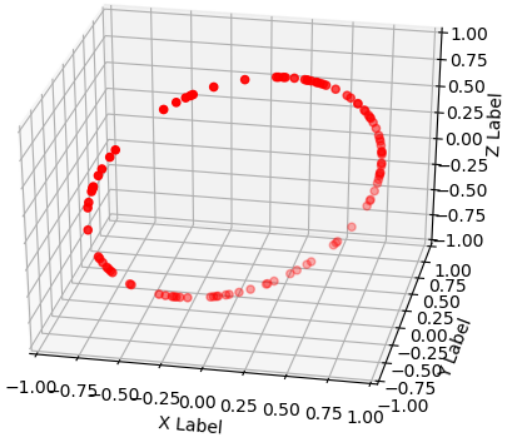

Fig. 3.15 The sample data distribution (100 data points)

Now construct its covariance matrix based on those 100 data points:

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i{}^T = \begin{bmatrix} 0.04082141 & -0.02348145 & -0.02353936 \\ -0.02348145 & 0.07436923 & -0.03918983 \\ -0.02353936 & -0.03918983 & 0.07597433 \end{bmatrix}$$

The data used to generate the covariance matrix can be found in Appendix. Noted that since all the distribution of all data are already centered at origin and the shape of the

distribution is a ring, there is no need to do mean adjustment or standard deviation adjustment. For 100 data points, the cluster center may not coincide with origin. If the mean adjustment is carries out, the result will not be right.

By solving $\widetilde{\boldsymbol{\Sigma}}$, its eigenvectors U and eigenvalues λ are obtained:

$$U = \begin{bmatrix} 6.40488723\text{e}-01 & 4.18040389\text{e}-01 & 6.44217687\text{e}-01 \\ 5.39476209\text{e}-01 & 3.52110563\text{e}-01 & -7.64842187\text{e}-01 \\ 5.46570778\text{e}-01 & -8.37412912\text{e}-01 & -3.82417659\text{e}-16 \end{bmatrix}$$

$$\lambda = [5.84614432\text{e}-01, 4.25486578\text{e}-01, 2.22044605\text{e}-16]$$

The eigenvector corresponds to the smallest eigenvalue is discarded as the intention is to compress the data to 2-dimensional. The largest 2 eigenvectors are plotted on the same plot:

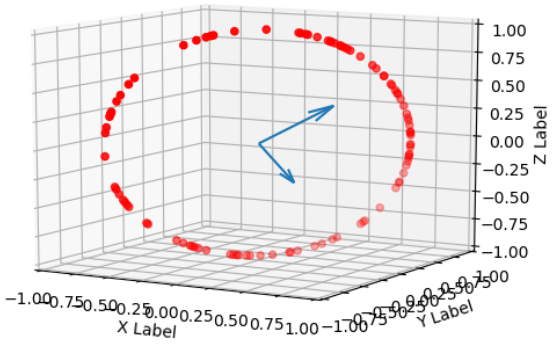

Fig. 3.16(a) The largest 2 eigenvectors with sample data view 1

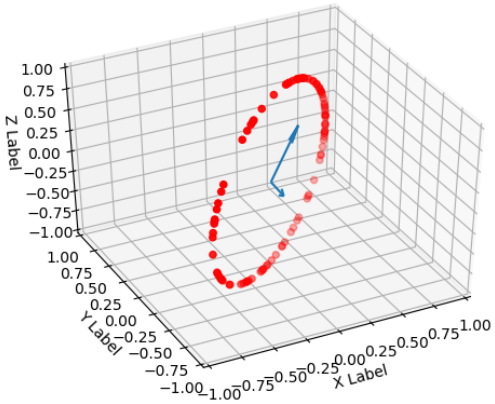

Fig. 3.16(b) The largest 2 eigenvectors with sample data view 2

Evidently, the two eigenvectors are in the same plane as the ring of data. This means that it only takes these 2 eigenvectors as the basis vector to represent all the data follow such distribution.

However, one critical issue with this approach is that PCA does this transformation by projection instead of rotation. Basically, the coordinates of each data point in the new coordinate system is obtained by the inner product of the point's coordinate vector and the

selected principal component vectors. For example, for point [0.44162798, 0.37197811, 0.81645393], its new coordinate is

$$[0.44162798 \quad 0.37197811 \quad 0.81645393]\begin{bmatrix} 6.40488723\text{e}-01 \\ 5.39476209\text{e}-01 \\ 5.46570778\text{e}-01 \end{bmatrix} = 0.9298$$

$$[0.44162798 \quad 0.37197811 \quad 0.81645393]\begin{bmatrix} 4.18040389\text{e}-01 \\ 3.52110563\text{e}-01 \\ -8.37412912\text{e}-01 \end{bmatrix} = -0.3681$$

i.e. [0.9298, -0.3681]. Geometrically, this is to project the data point in 3-d space onto that 2-d space spanned by the 2 eigenvectors. Such an operation is irreversible and therefore cannot be represented by a unitary matrix. In conclusion, PCA is not suitable for our task.

Now ask the second question is to be asked: can the constrained optimization technique we developed be used for other purpose which are originally dealt by PCA?

To reduce the dimensionality of some data points (which are not necessarily quantum data), assume they still live on some arbitrary plane but does not have any other pattern other than that, is it still possible to use our modified loss function:

$$L = \sum([w_{11} \quad w_{12} \quad w_{13}]V_{in})^2 + \lambda_1((w_1 * w_2)^2 + (w_1 * w_3)^2 + (w_2 * w_3)^2) + \lambda_2(\left(||w_1||^2 - 1\right)^2 + \left(||w_2||^2 - 1\right)^2 + \left(||w_3||^2 - 1\right)^2)$$

As mentioned before, those constraints guarantee that the norms of basis vectors and the mutual orthogonality between all basis vectors are preserved. Therefore, not just unit vectors but any arbitrary vectors in the vector space will have their norms and relative positions preserved, including the current vectors of interest. Of course, for the cases in which the plane of distribution does not intersect with the origin, mean adjustment needs to be done first.

However, for more general cases in which the distribution is not inherently 2-dimentional, the modification is non-trivial. PCA is capable to choosing the projection that minimize the loss of information. More investigations and studies need to be done for further discussion.

3.2 Quantum Denoising Autoencoder

3.2.1 Reproduce and Generalize the Results

In the original work of D. Bondarenko and P. Feldmann, the performance of QDAE was tested on GHZ state with x-flip noise. Here, we would like to reproduce the result and further developed the idea into testing other types of quantum states and flipping noise.

Tool learnt and used: MATLAB
Operating system used: Ubuntu

MATLAB is one of the most widely used simulation/computational software used in the academia across various of fields. Ubuntu is a free and open-source Linux distribution. The following results are all coded in MATLAB, compiled in Linux operating system (Ubuntu) and executed on the supercomputer from National Supercomputing Centre Singapore (NSCC). It took several months in total to generate all the results below.

The architecture of QDAR is set to $[m, 1, m]$, which means m nodes on input layer, 1 node in 1-layer hidden layer and m nodes again on the output layer. The number of nodes m is the same as the size of the original input quantum state. If the inputs are qubits, $m = 2$; if the inputs are qutrits, $m = 3$, etc. The hidden layer itself is the bottleneck which forces a simpler hidden representation with latent variables of lower dimension to be found, which is used to find the common features in all of the inputs and use it as the 'uncorrupted data'. On top of what was mentioned in Section 2.4.4, we use y-flip and z-

flip noise beside x-flip noise to corrupt the data - GHZ states of qunits for n =2,3,4,5. y-flip operator is defined as

$$\sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

and z-flip operator is defined as

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Together, σ_z, σ_y, σ_z are called Pauli gates.

Input layer size and output layer size m varies according to the size of the input quantum states ($m = n$).

Other hyperparameter settings are as follows:

Batch size = 500
It means for every round of training there are have 500 input state vectors corrupted by the same type of noise with the same probability of being corrupted.

Number of iterations = 100
Each round of training undergoes 100 iterations of updating the weights/unitary matrix.

Target values and the data corrupted by the same type of flipping noise.
After 100 iterations, the average fidelity of the 500 output (supposedly denoised state vectors) is computed as the benchmark and the standard deviation of the 500 fidelities to plot the error bar.

GHZ State

The fidelity versus the probability of flipping (in percentage) varies from 0 to 1 with 0.01 increment is plotted. The probability of flipping is the same as defined in Section 2.4.4. It dictates the fidelities of the input states/how noisy the input is. For all cases, the fidelities follow the following pattern:

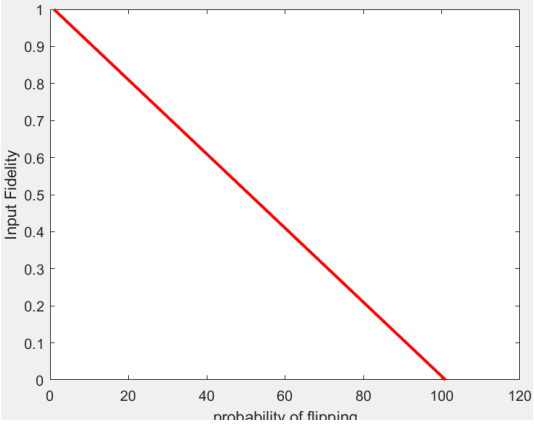

Fig. 3.17 GHZ-State Input Fidelity versus Probability of Flipping Plot

Intuitively, as the probability of the occurrence of flipping noise gets higher, regardless of which type of flipping noise (x, y or z), the input fidelity gets lower. When the probability is 0%, the fidelity is 1; when the probability is 100%, the fidelity is 0. Anything in between ideally can be interpolated as a straight-line segment.

The results are shown below:

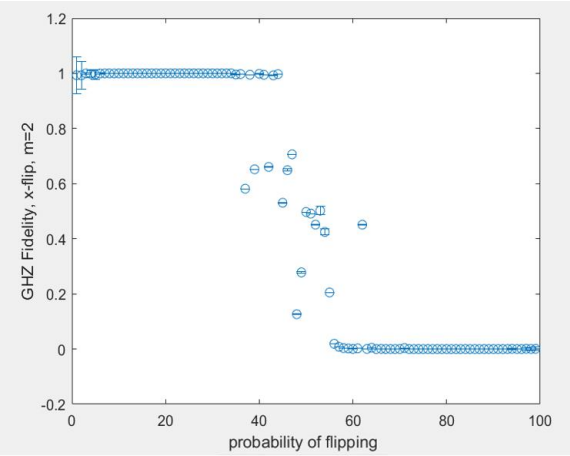

Fig. 3.18(a) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, x-flip)

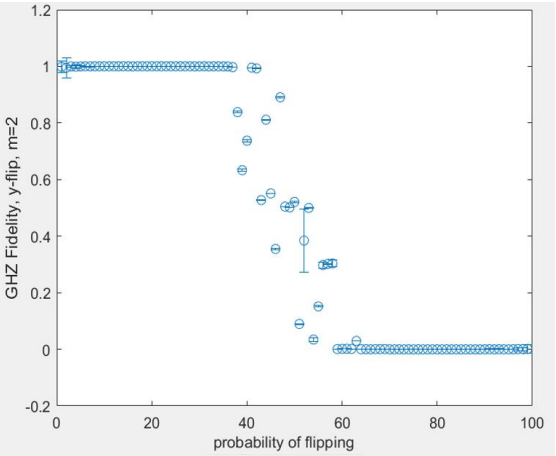

Fig. 3.18(b) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, y-flip)

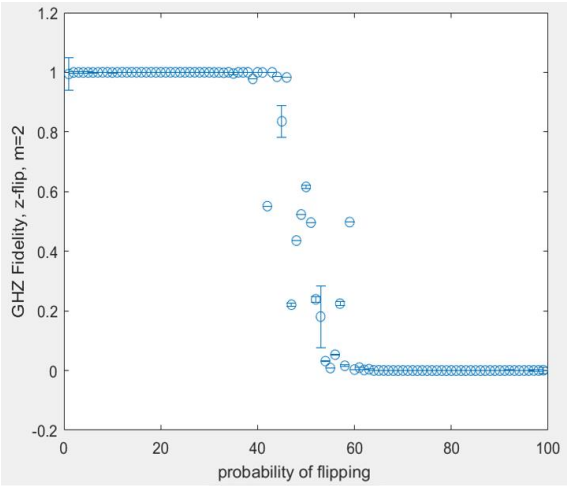

Fig. 3.18(c) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, z-flip)

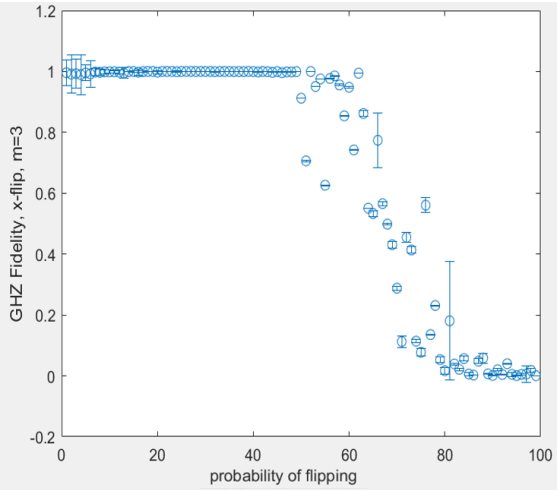

Fig. 3.18(d) GHZ-State Fidelity versus Probability of Flipping Plot (m=3, x-flip)

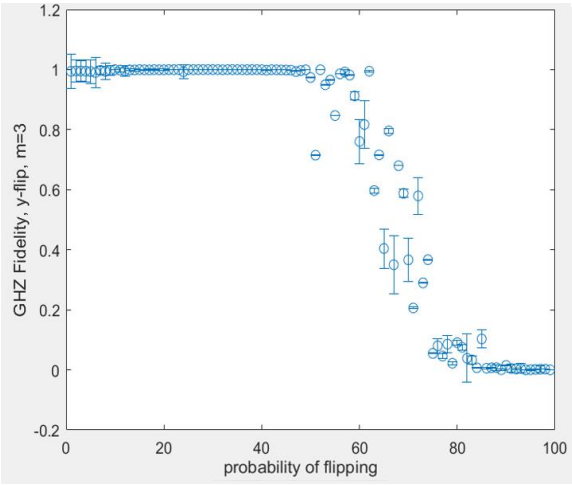

Fig. 3.18(e) GHZ-State Fidelity versus Probability of Flipping Plot (m=3, y-flip)

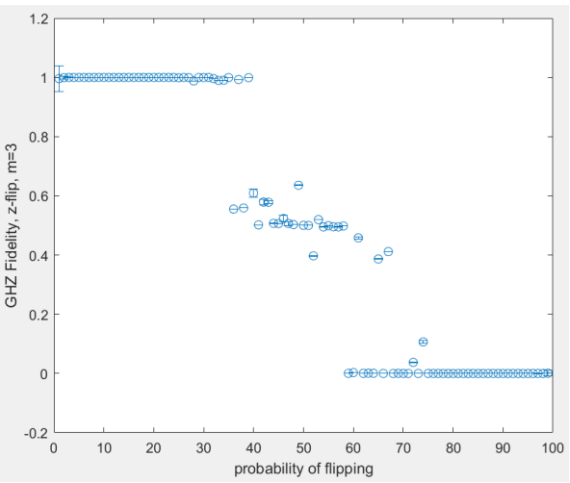

Fig. 3.18(f) GHZ-State Fidelity versus Probability of Flipping Plot (m=3, z-flip)

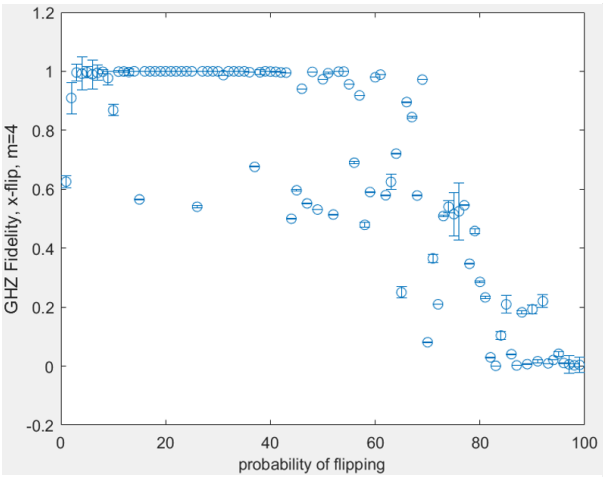

Fig. 3.18(g) GHZ-State Fidelity versus Probability of Flipping Plot (m=4, x-flip)

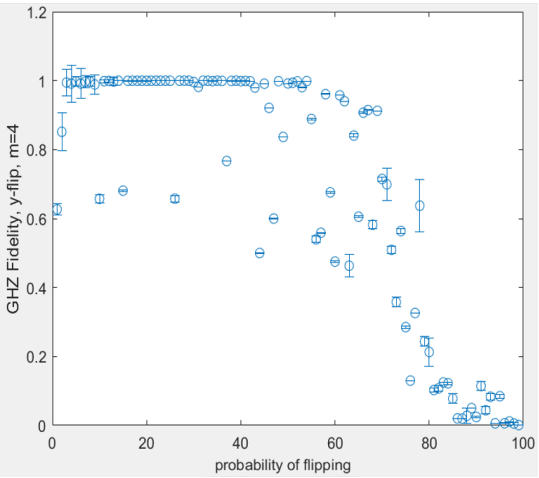

Fig. 3.18(h) GHZ-State Fidelity versus Probability of Flipping Plot (m=4, y-flip)

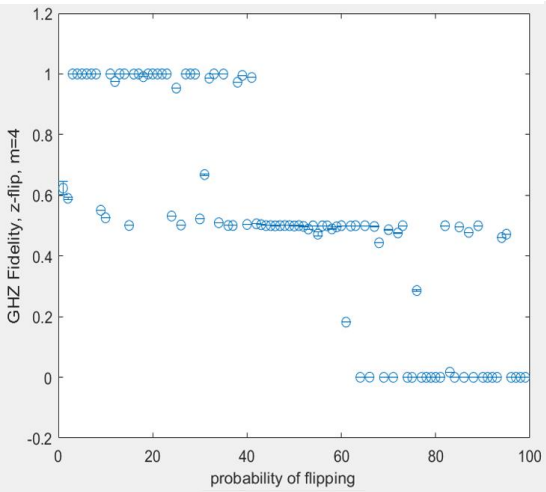

Fig. 3.18(i) GHZ-State Fidelity versus Probability of Flipping Plot (m=4, z-flip)

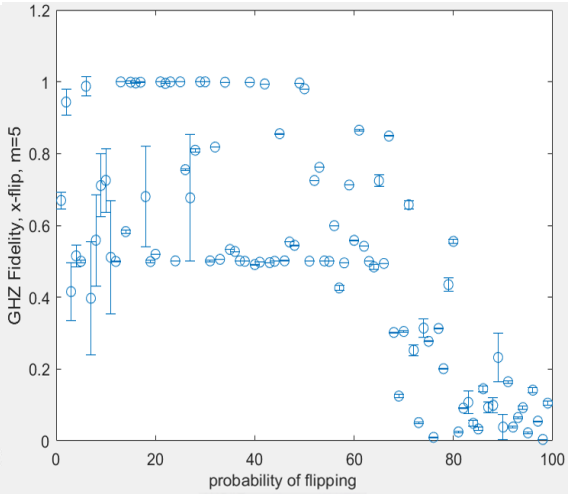

Fig. 3.18(j) GHZ-State Fidelity versus Probability of Flipping Plot (m=5, x-flip)

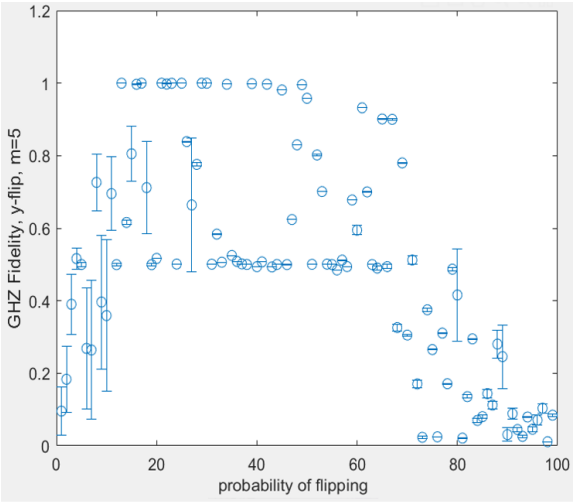

Fig. 3.18(k) GHZ-State Fidelity versus Probability of Flipping Plot (m=5, y-flip)

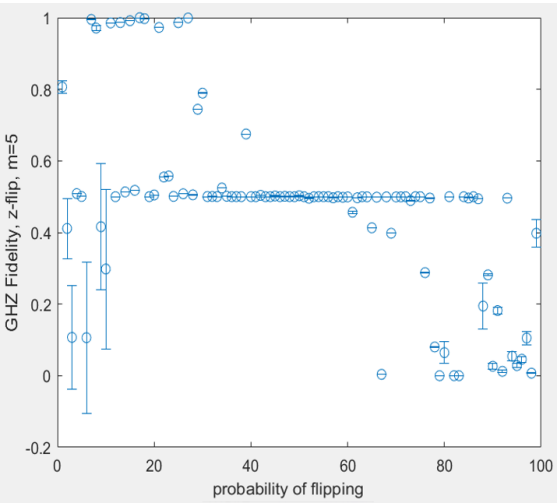

Fig. 3.18(l) GHZ-State Fidelity versus Probability of Flipping Plot (m=5, z-flip)

Observation 1: for m = 2, the denoising ability started to drop for all three types of flipping noise when the flipping probability is a bit more than 40%. That is basically the limit for this QDAE being functional under the current hyperparameter settings. Within the limit, the performance of QDAE is rather stable - almost all denoised outputs have fidelities of 1 with very little variance. From 40% to 60% flipping probability, there seems to have a

transitional interval, in which the output fidelity gets decreased very quickly while the output fidelities have large variance. Beyond 60% flipping probability, the output fidelities are stably 0.

Observation 2: for m =3, the limit for QDAE to be functional seems increased by some margin (increased to 50% for x-flip and y-flip). This is more prominent for x-flip and y-flip. The threshold after which the output fidelity becomes completely 0 also gets pushed back somewhat. However, there are notable inconsistencies of the denoising performance when the input fidelity is very high (roughly 0% to 5% flipping probability).

Observation 3: for m=4, the denoising performance becomes unstable even within the functional limit. Notable variance of output fidelities is similar to that for m=3 case when input fidelity is very high.

Observation 4: for m=5, high variance of output fidelity is observed within the functional range. QDAE within the currently hyperparameter settings becomes unreliable.

Observation 5: As m gets larger, the variance of the output fidelities when noise rate is very low becomes more and more prominent. Observe the loss-versus-iteration plot of those high variance points,

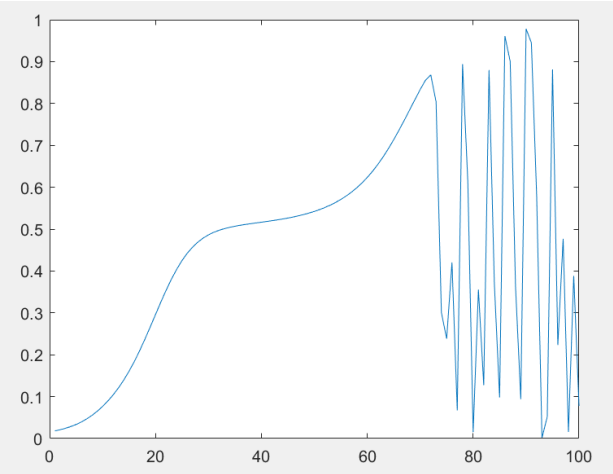

Fig. 3.19 GHZ-State Fidelity versus no. of Iteration Plot (m=5, x-flip, probability of flipping = 0%)

it can be seen that the loss gets haywire after reaching a certain value. It could be the case that 100 iteration is considered overtraining for such a setting with low noise and cause the gradient explosion, which in turn gives unstable denoising performance.

Now test the QDAE of the same architecture on other types of quantum states corrupted by the same types of noises. All hyperparameter settings remain the same.

W State

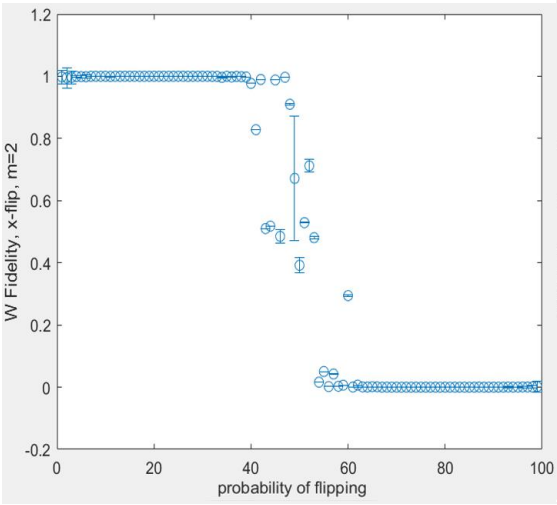

Fig. 3.20(a) W-State Fidelity versus Probability of Flipping Plot (m=2, x-flip)

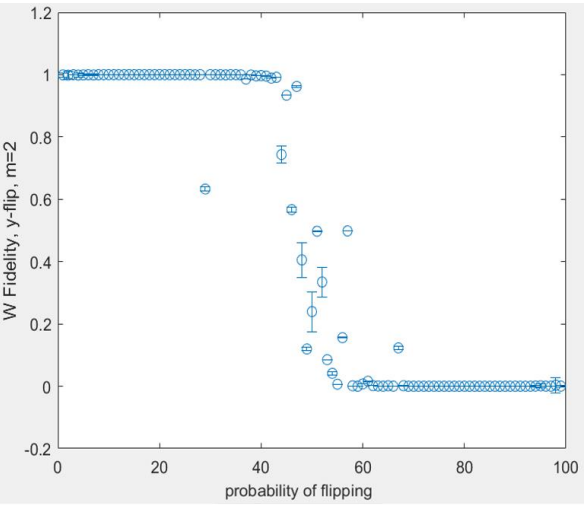

Fig. 3.20(b) W-State Fidelity versus Probability of Flipping Plot (m=2, y-flip)

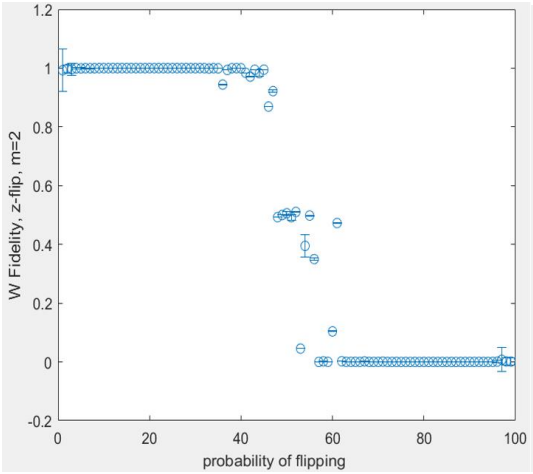

Fig. 3.20(c) W-State Fidelity versus Probability of Flipping Plot (m=2, z-flip)

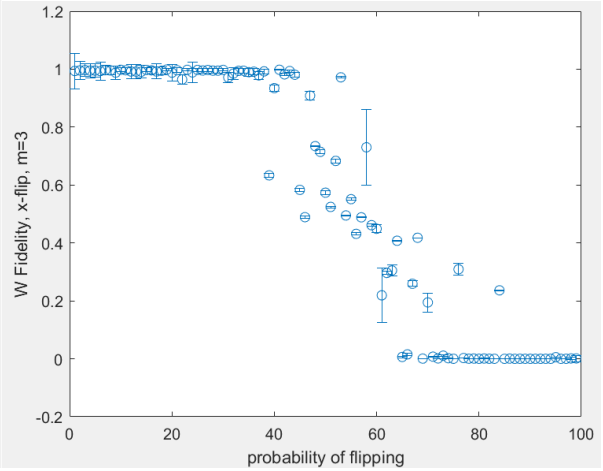

Fig. 3.20(d) W-State Fidelity versus Probability of Flipping Plot (m=3, x-flip)

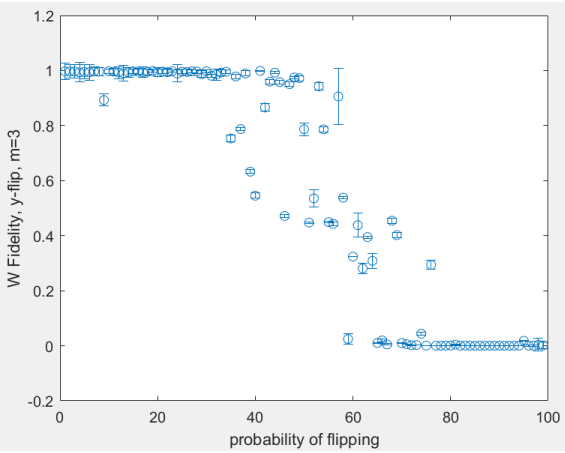

Fig. 3.20(e) W-State Fidelity versus Probability of Flipping Plot (m=3, y-flip)

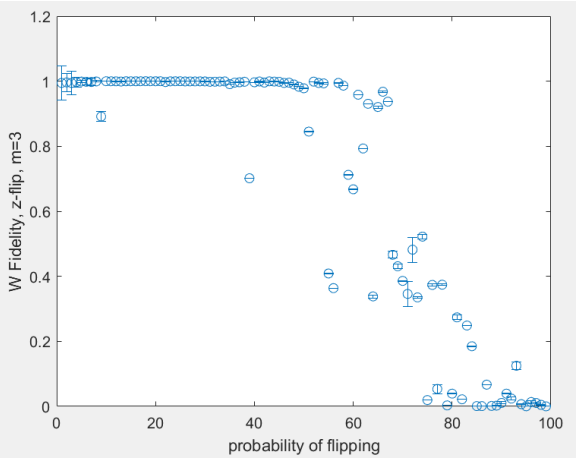

Fig. 3.20(f) W-State Fidelity versus Probability of Flipping Plot (m=3, z-flip)

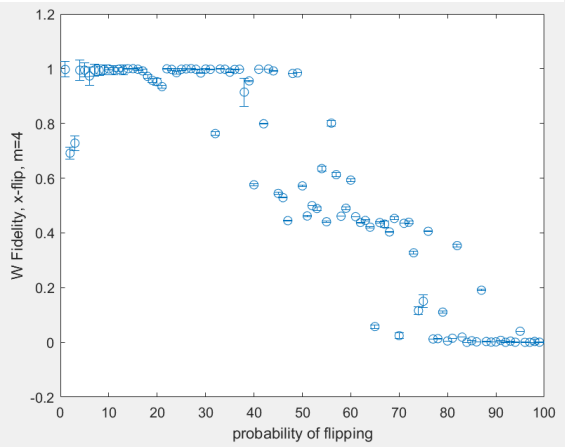

Fig. 3.20(g) W-State Fidelity versus Probability of Flipping Plot (m=4, x-flip)

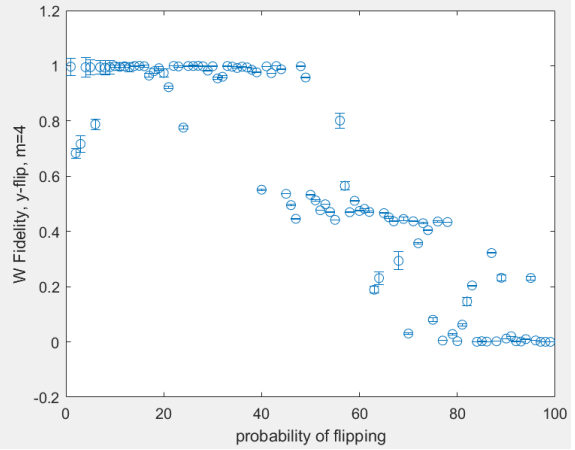

Fig. 3.20(h) W-State Fidelity versus Probability of Flipping Plot (m=4, y-flip)

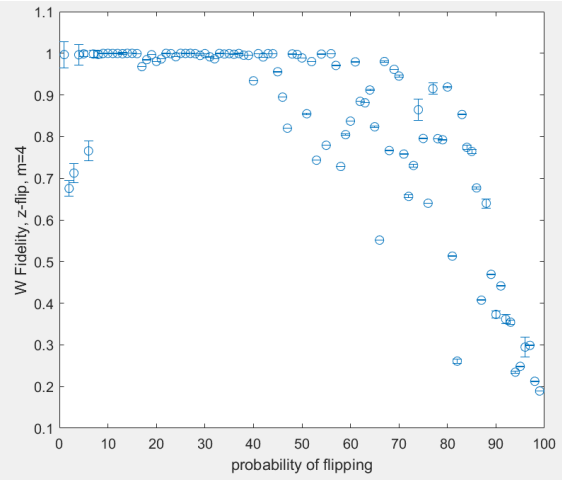

Fig. 3.20(i) W-State Fidelity versus Probability of Flipping Plot (m=4, z-flip)

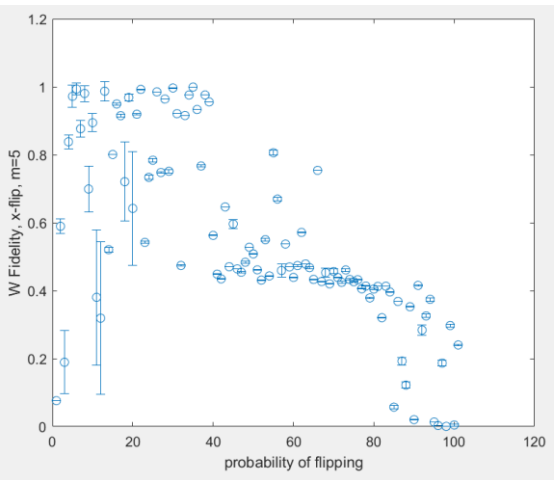

Fig. 3.20(j) W-State Fidelity versus Probability of Flipping Plot (m=5, x-flip)

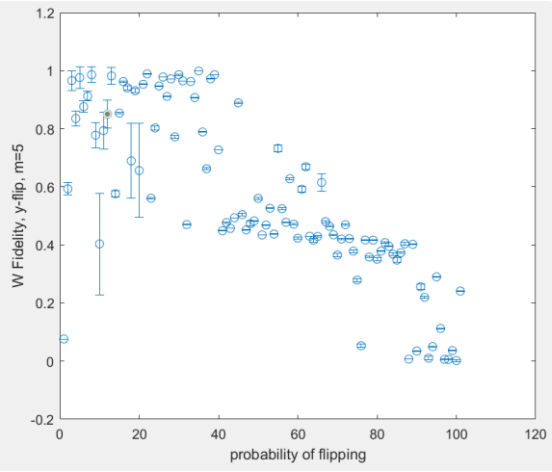

Fig. 3.20(k) W-State Fidelity versus Probability of Flipping Plot (m=5, y-flip)

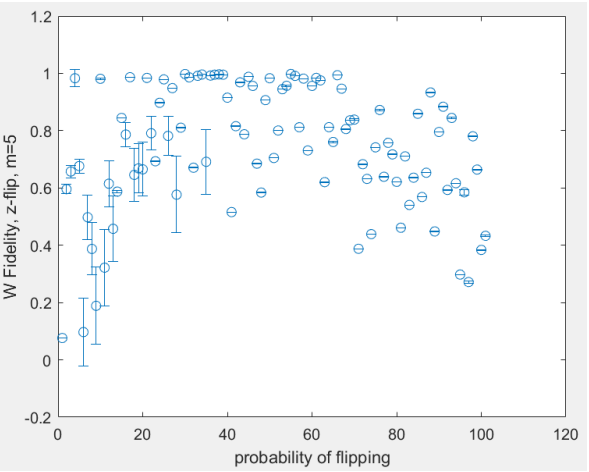

Fig. 3.20(l) W-State Fidelity versus Probability of Flipping Plot (m=5, z-flip)

Observation: The overall pattern and behavior of W state with respect to different m and noise is very similar to that of GHZ state. The only notable difference is for m=5 z-flip, the there is no transitional interval and the output fidelity did not decrease all the way to 0. It only varies a lot for different flipping probabilities without clear pattern. Based on its loss-versus-iteration plot:

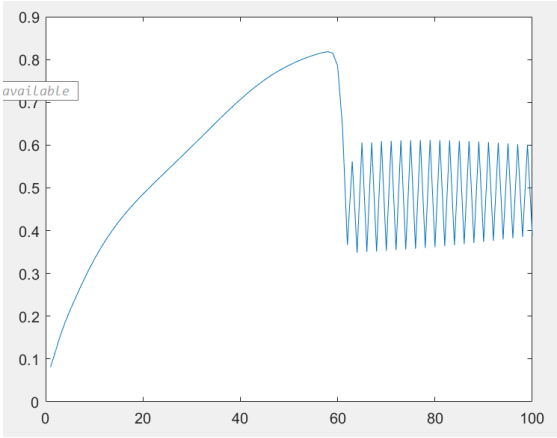

Fig. 3.21(a) W-State Fidelity versus no. of Iteration Plot (m=5, z-flip, probability of flipping = 0%)

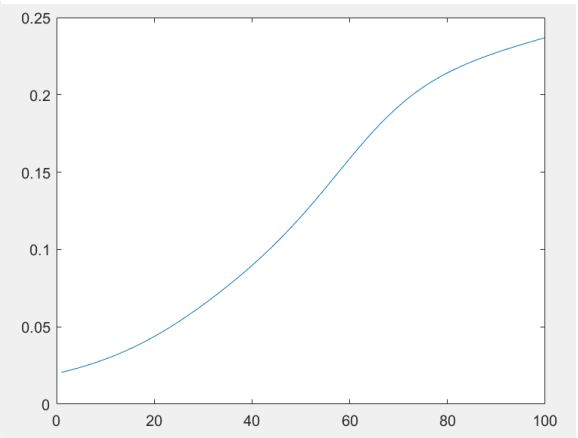

Fig. 3.21(b) W-State Fidelity versus no. of Iteration Plot (m=5, z-flip, probability of flipping = 100%)

Evidently, when iteration=100, low noise case is over-trained and high noise case is not.

Based on the two plots and with the GHZ state loss-versus-iteration plot, an educated guess can be made: over-training causes high variance in denoised data.

Zero State

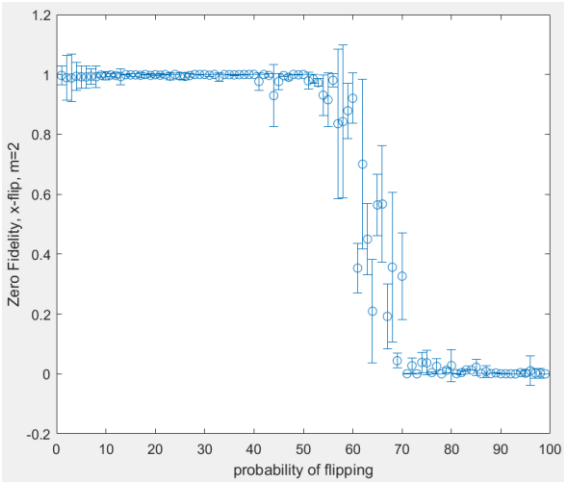

Fig. 3.22(a) Zero-State Fidelity versus Probability of Flipping Plot (m=2, x-flip)

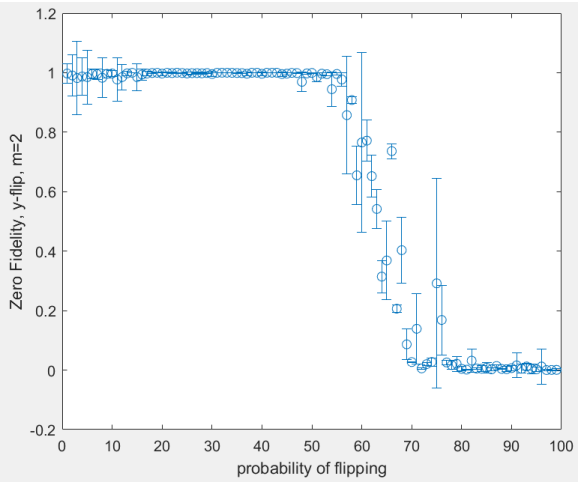

Fig. 3.22(b) Zero-State Fidelity versus Probability of Flipping Plot (m=2, y-flip)

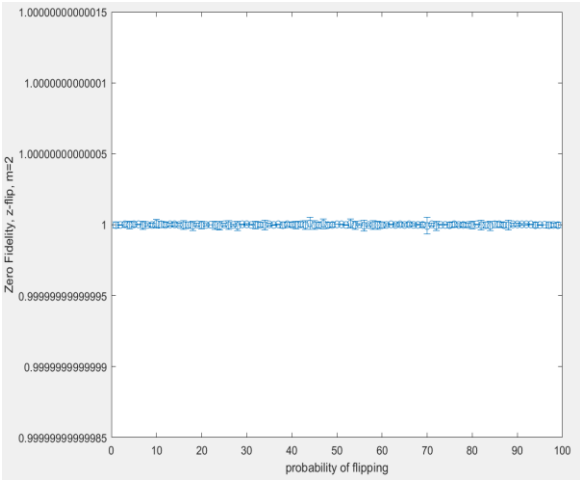

Fig. 3.22(c) Zero-State Fidelity versus Probability of Flipping Plot (m=2, z-flip)

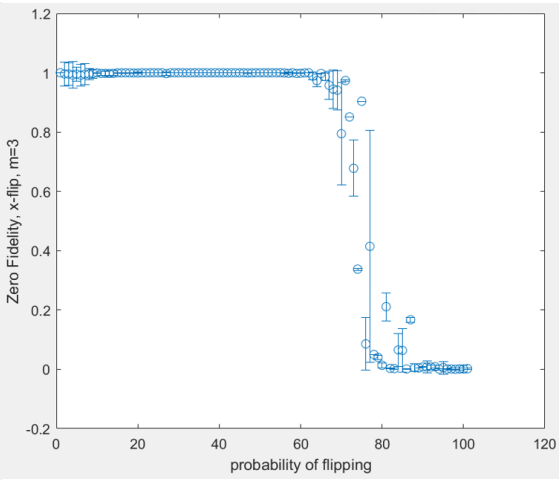

Fig. 3.22(d) Zero-State Fidelity versus Probability of Flipping Plot (m=3, x-flip)

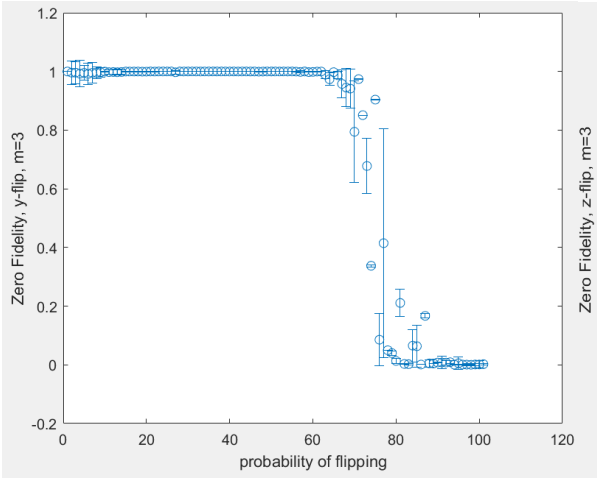

Fig. 3.22(e) Zero-State Fidelity versus Probability of Flipping Plot (m=3, y-flip)

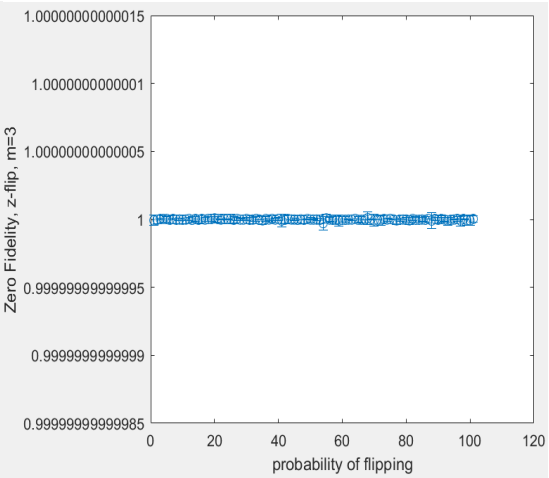

Fig. 3.22(f) Zero-State Fidelity versus Probability of Flipping Plot (m=3, z-flip)

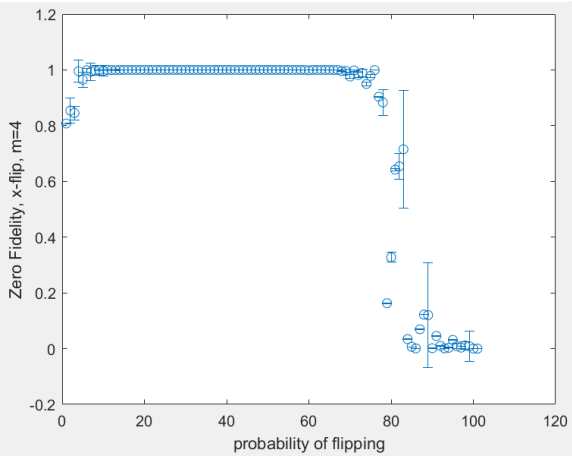

Fig. 3.22(g) Zero-State Fidelity versus Probability of Flipping Plot (m=4, x-flip)

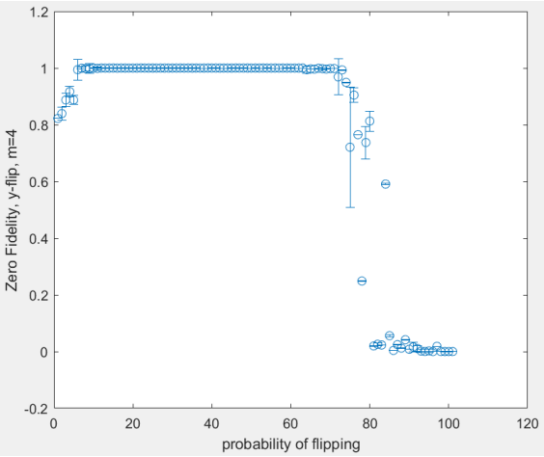

Fig. 3.22(h) Zero-State Fidelity versus Probability of Flipping Plot (m=4, y-flip)

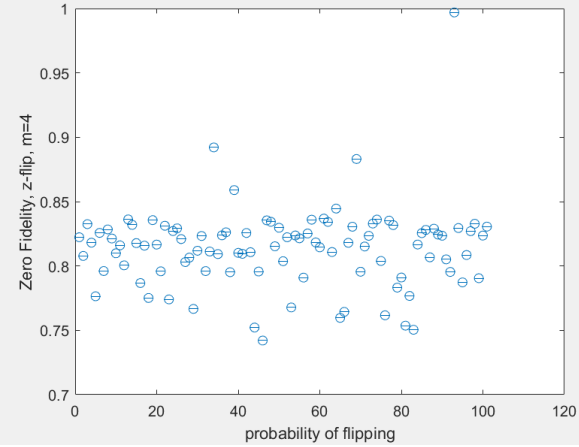

Fig. 3.22(i) Zero-State Fidelity versus Probability of Flipping Plot (m=5, z-flip)

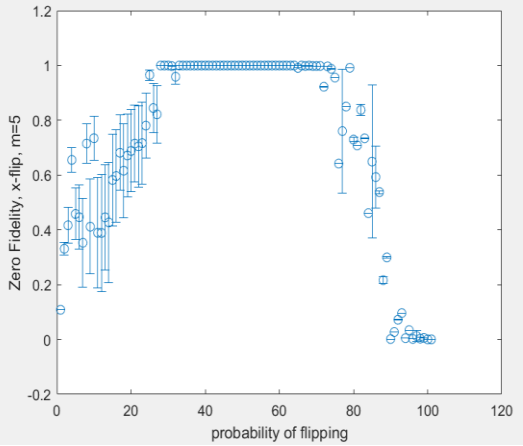

Fig. 3.22(j) Zero-State Fidelity versus Probability of Flipping Plot (m=5, x-flip)

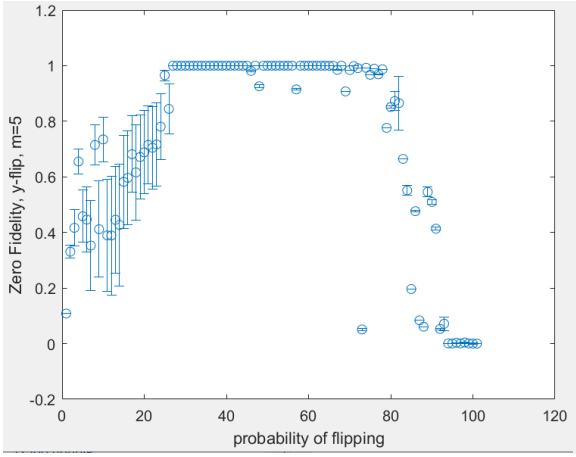

Fig. 3.22(k) Zero-State Fidelity versus Probability of Flipping Plot (m=5, y-flip)

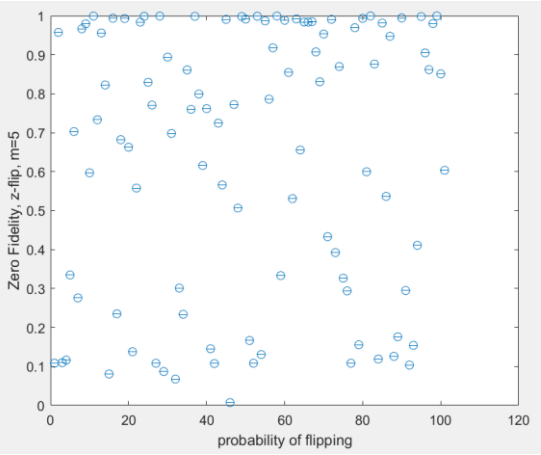

Fig. 3.22(l) Zero-State Fidelity versus Probability of Flipping Plot (m=5, z-flip)

Observation 1: notice that for m=2 and m=3 z-flip noise, the fidelities are always almost 1 throughout. Their input states fidelities with respect to the probability of flipping are shown as below:

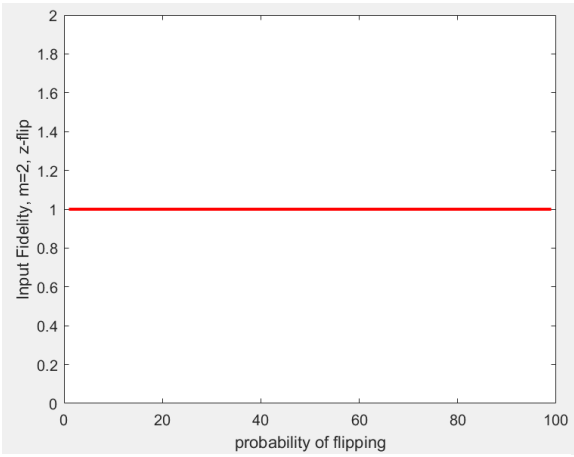

Fig. 3.23(a) Zero-State Input Fidelity versus Probability of Flipping Plot (m=2, z-flip)

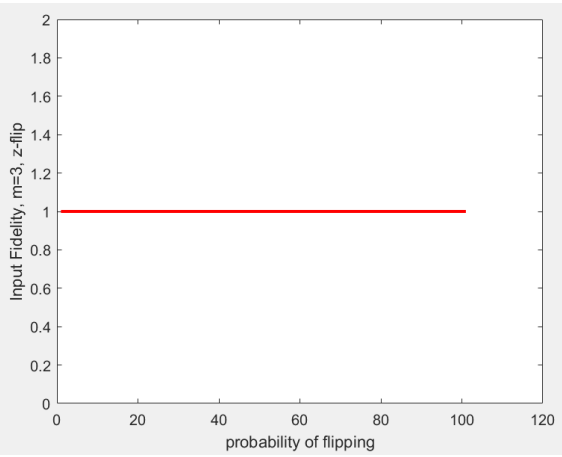

Fig. 3.23(b) Zero-State Input Fidelity versus Probability of Flipping Plot (m=3, z-flip)

Their input states fidelity is also 1 throught. The reason behind is actually that zero states are immune to z-flip transformation. This fact can be seen from the following induction:

$$\sigma_z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

No matter the z-flip gate acts on which part of the zero state, it will not have effect:

$$(I^{\otimes i} \otimes \sigma_z \otimes I^{\otimes j})|0\rangle^n = (I|0\rangle) \otimes (I|0\rangle) \otimes \ldots \otimes (\sigma_z|0\rangle \otimes \ldots \otimes (I|0\rangle) = |0\rangle$$

This is true for qunit zero state as well for any positive integer n.

However, for m=4 case, the fidelities of the output states for z-flip error are non-trivially deviated from 1 (Fig. 3.23(i)) and fluctuated about somewhere in between 0.8 and 0.85. What is even worse, for m=5, the output fidelity seems entirely random regardless of the flipping probability (Fig. 3.23(l)). Observe the loss-versus-iteration plot again for these two states:

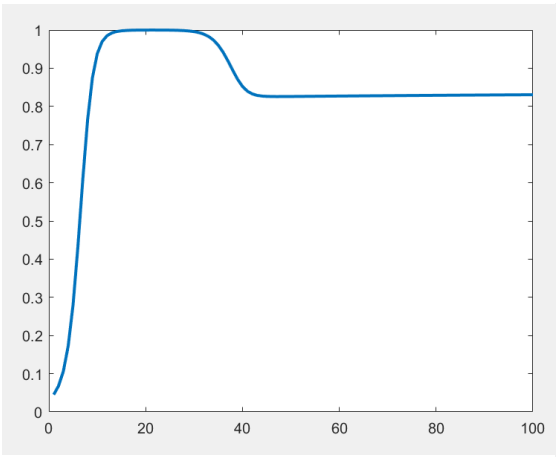

Fig. 3.24(a) Zero-State Fidelity versus no. of Iteration Plot (m=4, z-flip, probability of flipping = 100%)

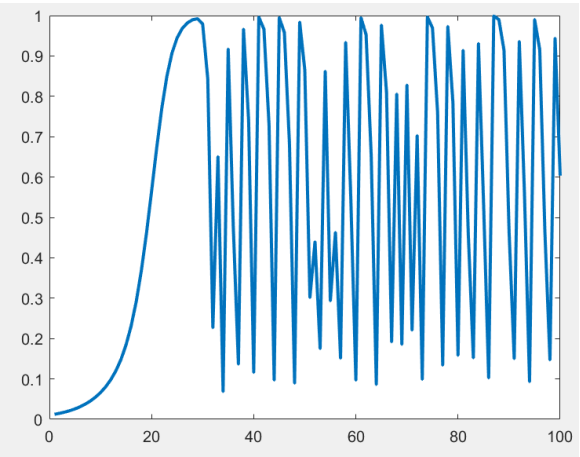

Fig. 3.24(b) Zero-State Fidelity versus no. of Iteration Plot (m=5, z-flip, probability of flipping = 100%)

For m=4 case, the points are distributed around the value slightly more than 0.8. This indeed corresponds to the fact that the output fidelities for m=4 are centered around somewhere betwwen 0.8 and 0.85 (with some fluctuation for different flipping probabilities). As for m=5 case, again the loss went haywire, which corresponds to the chaotic scattered plot.

Observation 2: for x-flip and y-flip, QDAE on Zero state seems outperformed that on GHZ state and W state. When m = 2, the functional limit for Zero state case is between 50% and 60%, whereas for GHZ state case and W state case, the funcional limit is about 40%. When m=3, the functional limit for Zero state is around 70% (further increased compared to m=2 case), whereas for GHZ state and W state it is about 50%. For m=4 and m=5 case, the functional limit even increased to around 80%, which is also higher than that of GHZ state and W state.

Observation 3: the larger variance as m gets larger for small flipping probability states also appeared. By obersving the loss-versus-iteratio plot:

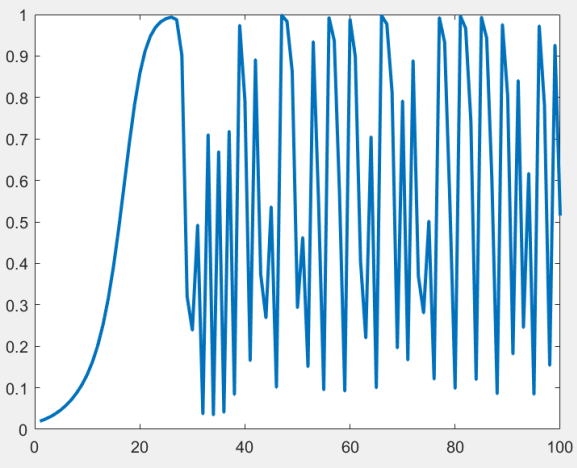

Fig. 3.25 Zero-State Fidelity versus no. of Iteration Plot (m=5, z-flip, probability of flipping = 25%)

Indeed, it was over-trained again. This result further verified our educated guess.

Plus State

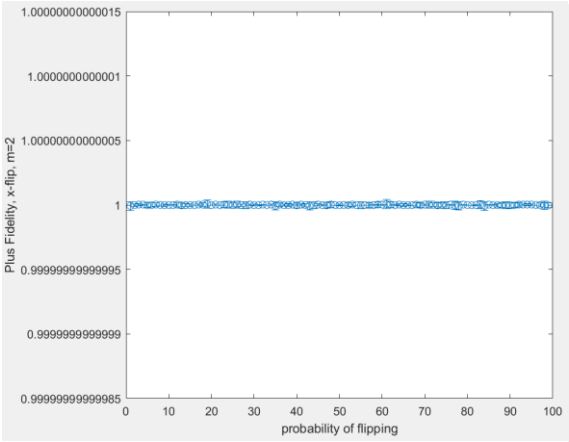

Fig. 3.26(a) Plus-State Fidelity versus Probability of Flipping Plot (m=2, x-flip)

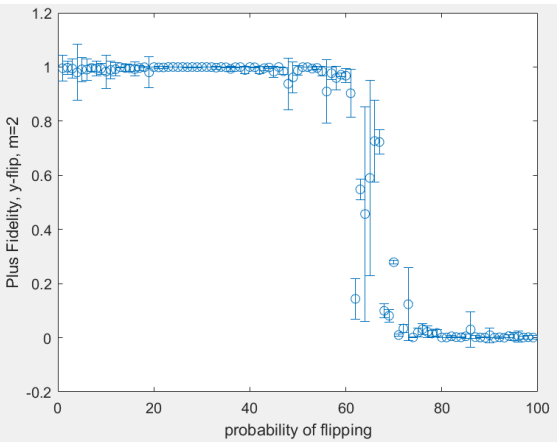

Fig. 3.26(b) Plus-State Fidelity versus Probability of Flipping Plot (m=2, y-flip)

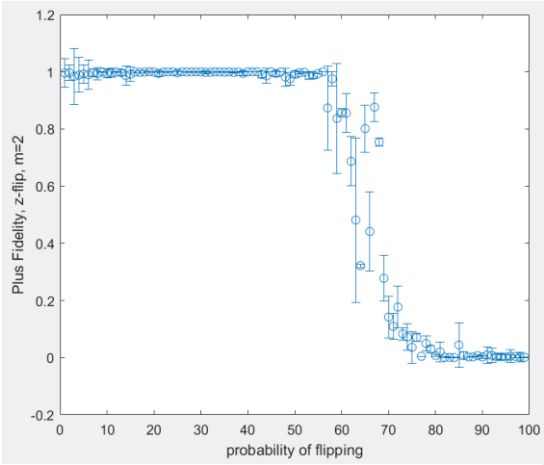

Fig. 3.26(c) Plus-State Fidelity versus Probability of Flipping Plot (m=2, z-flip)

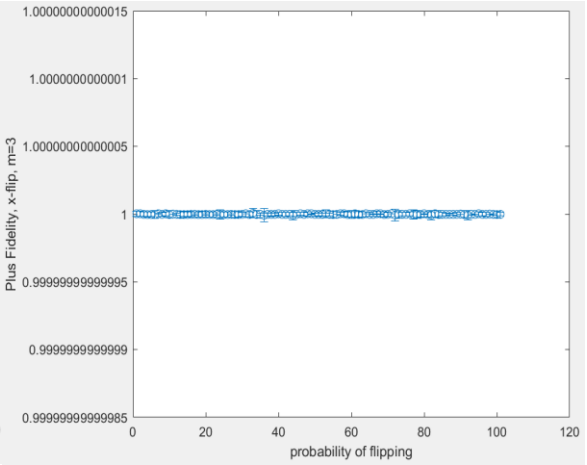

Fig. 3.26(d) Plus-State Fidelity versus Probability of Flipping Plot (m=3, x-flip)

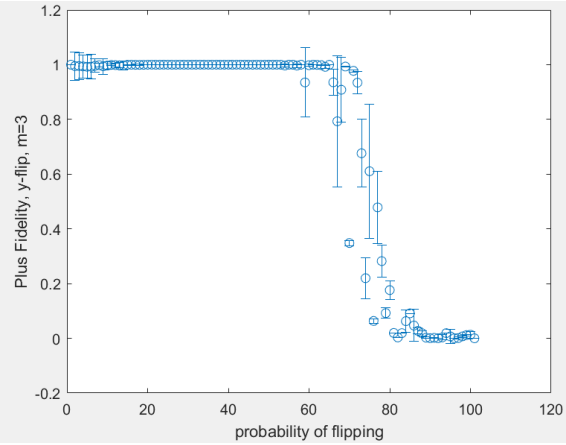

Fig. 3.26(e) Plus-State Fidelity versus Probability of Flipping Plot (m=3, y-flip)

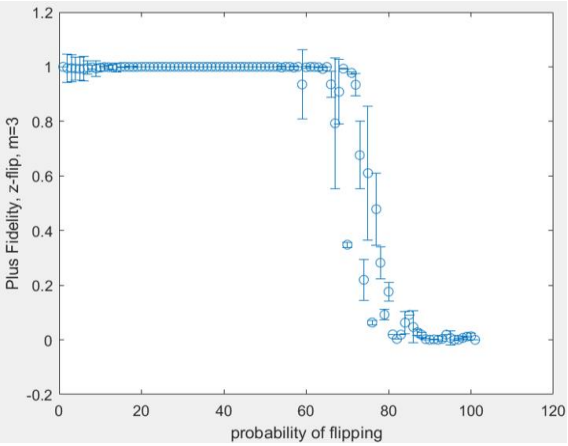

Fig. 3.26(f) Plus-State Fidelity versus Probability of Flipping Plot (m=3, z-flip)

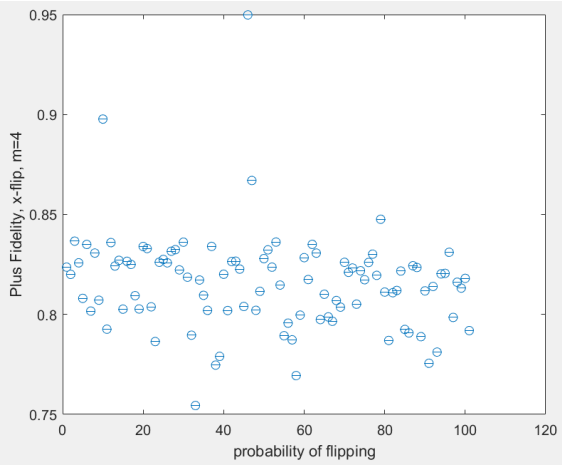

Fig. 3.26(g) Plus-State Fidelity versus Probability of Flipping Plot (m=4, x-flip)

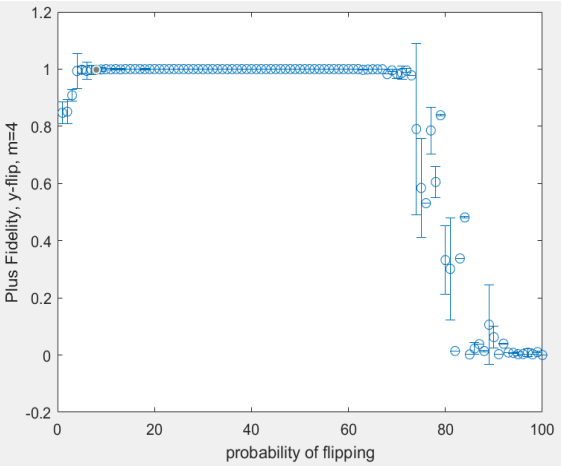

Fig. 3.26(h) Plus-State Fidelity versus Probability of Flipping Plot (m=4, y-flip)

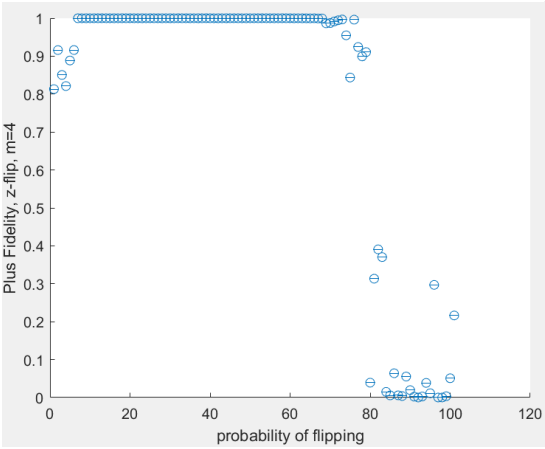

Fig. 3.26(i) Plus-State Fidelity versus Probability of Flipping Plot (m=4, z-flip)

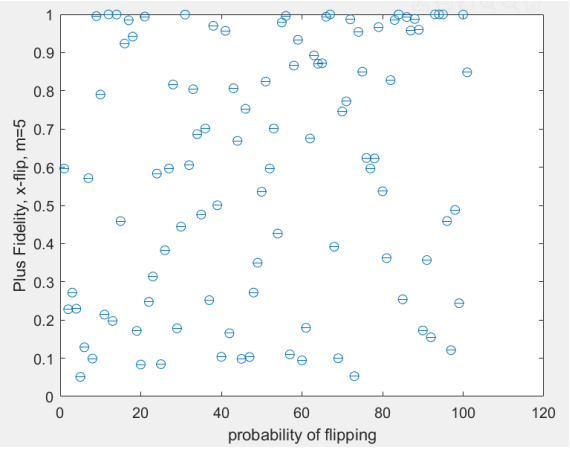

Fig. 3.26(j) Plus-State Fidelity versus Probability of Flipping Plot (m=5, x-flip)

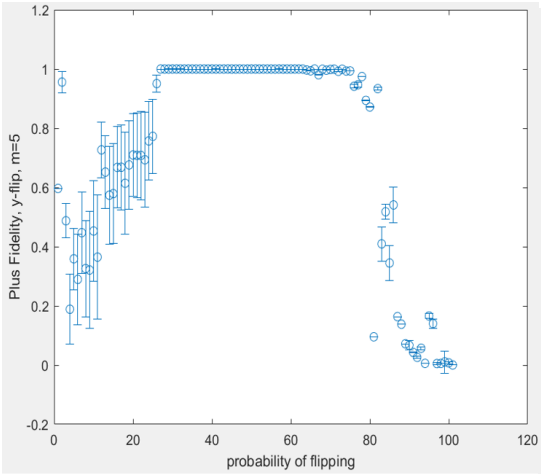

Fig. 3.26(k) Plus-State Fidelity versus Probability of Flipping Plot (m=5, y-flip)

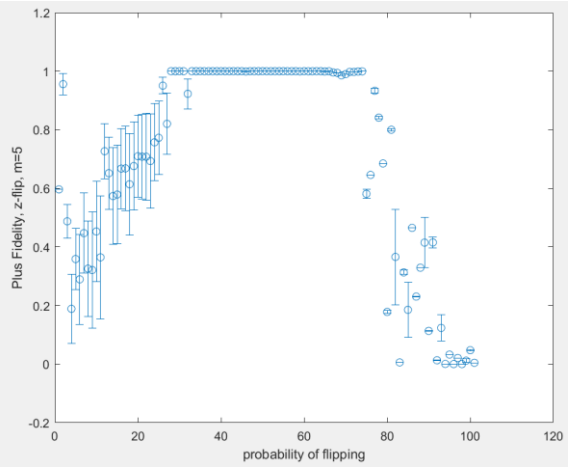

Fig. 3.26(l) Plus-State Fidelity versus Probability of Flipping Plot (m=5, z-flip)

Observation 1: Plus state plots behave very similar to that of zero state in every aspect, except that the constant fidelity plots are now for x-flip, m-2 and m=3 cases. This can be easily explained by the following induction:

$$\sigma_x|+\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = |+\rangle$$

and this is true regardless of which part of the state σ_x is acting on:

$$(I^{\otimes \mathrm{i}} \otimes \sigma_x \otimes I^{\otimes \mathrm{j}})|+\rangle^{\mathbf{n}} = (I^{\otimes \mathrm{i}} \otimes \sigma_x \otimes I^{\otimes \mathrm{j}})\frac{1}{\sqrt{n}}(|0\rangle + |1\rangle + \cdots + |n\rangle)$$
$$= (I^{\otimes \mathrm{i}} \otimes \sigma_z \otimes I^{\otimes \mathrm{j}})(|+\rangle \otimes |+\rangle \ldots \otimes |+\rangle)$$
$$= (I|+\rangle) \otimes (I|+\rangle) \otimes \ldots \otimes (\sigma_x|+\rangle \otimes \ldots \otimes (I|+\rangle) = |+\rangle$$

When m=4 the constant plots also became deviated and fluctuated. When reaches m=5, the plot becomes completely random as like the z-flip case for Zero state.

In terms of the variance, functional limit for each m, Plus state is almost indistinguishable from Zero state.

Now retrain plus state m=5 x-flip case with 16 iterations to see what happens under non-overtraining situation:

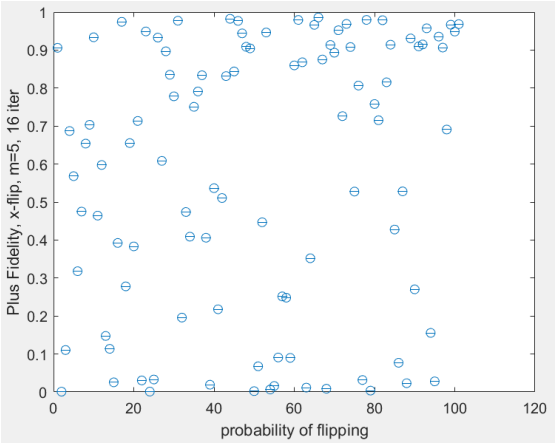

Fig. 3.27(a) Plus-State Fidelity versus Probability of Flipping Plot (m=5, x-flip, 16 iterations per training)

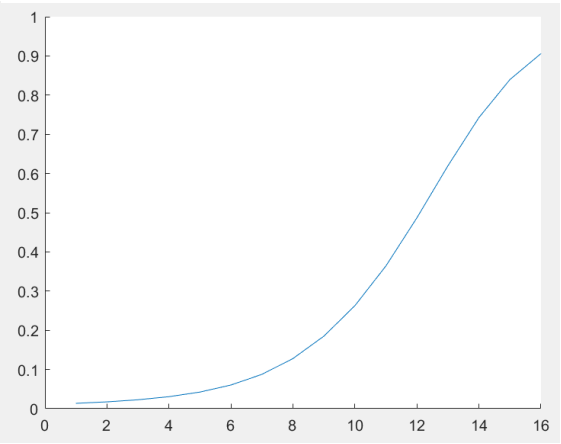

Fig. 3.27(b) Plus-State Fidelity versus no. of Iteration Plot (m=5, x-flip, probability of flipping = 0%)

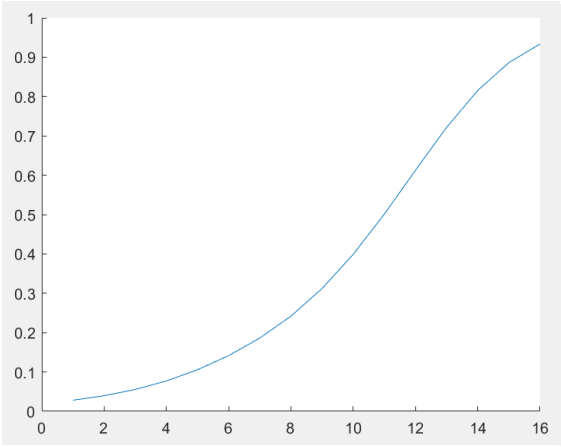

Fig. 3.27(c) Plus-State Fidelity versus no. of Iteration Plot (m=5, x-flip, probability of flipping = 10%)

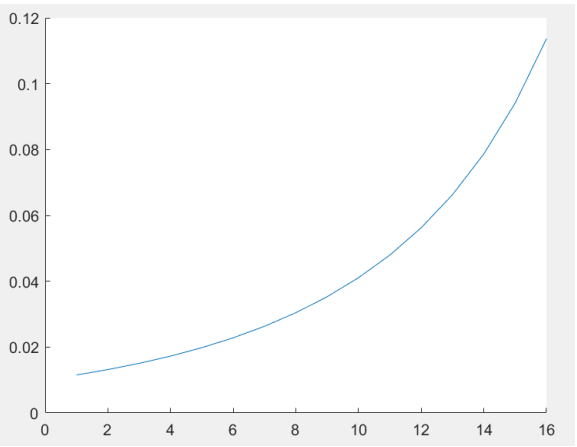

Fig. 3.27(d) Plus-State Fidelity versus no. of Iteration Plot (m=5, x-flip, probability of flipping = 14%)

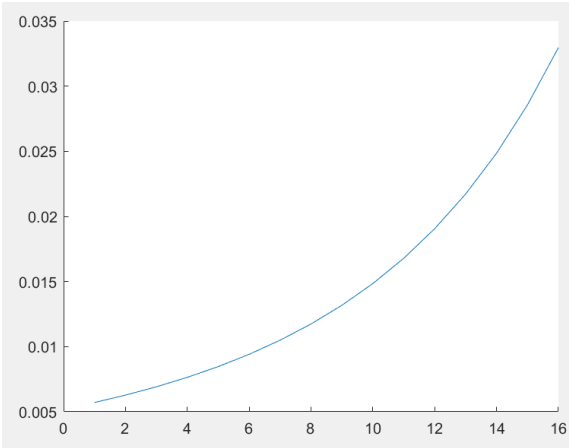

Fig. 3.27(e) Plus-State Fidelity versus no. of Iteration Plot (m=5, x-flip, probability of flipping = 25%)

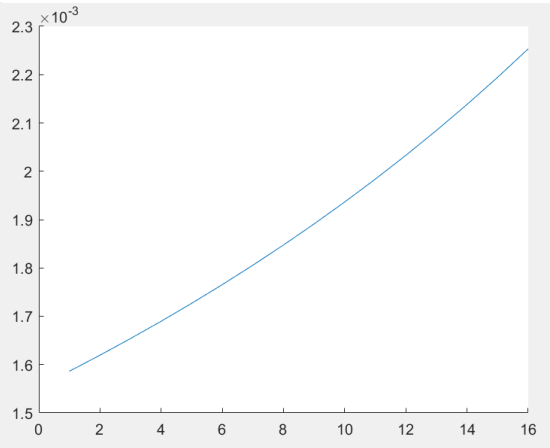

Fig. 3.27(f) Plus-State Fidelity versus no. of Iteration Plot (m=5, x-flip, probability of flipping = 50%)

The fidelity distribution is still random and without any pattern. By sampling some of the loss-versus-iteration plots it can be observed that none of the case is over-trained. Yet, the final loss for different flipping probabilities can vary a lot: some are near to one while others can be as low as in the magnitude of 10^{-3}. Therefore, it can be concluded that the randomness in the plot is caused by different accuracies for different flipping probabilities case. On the other hand, overtraining causes high variance within the same flipping probabilities (large error bar).

In conclusion, the DQAE with current architecture and hyperparameter setting works not only for GHZ state, but also for other states such as W state, Zero state and Plus state. The performance varies from states to states by certain extent. And as the parameter m increased, it works for inputs with lower signal-to-noise ratio (higher flipping probability). One caveat is that the learning rate and number of iterations should be adjusted adaptively to make sure that the training processes are conducted properly without over-training or under-straining case.

3.2.2 Same Training and Testing State with Different Training and Testing Noise

It is an interesting question to ask, whether the denoising works for the same state but with noise different from the one for training. It is interesting because one characteristic of machine learning is the ability to generalize. Therefore, we extend the idea into testing a few states with different testing noise from the training noise:

Based on the prior experiences from the previous section, set 50 iterations per training to avoid over-training. For each result, check the loss-versus-iteration plot to make sure the training was executed properly.

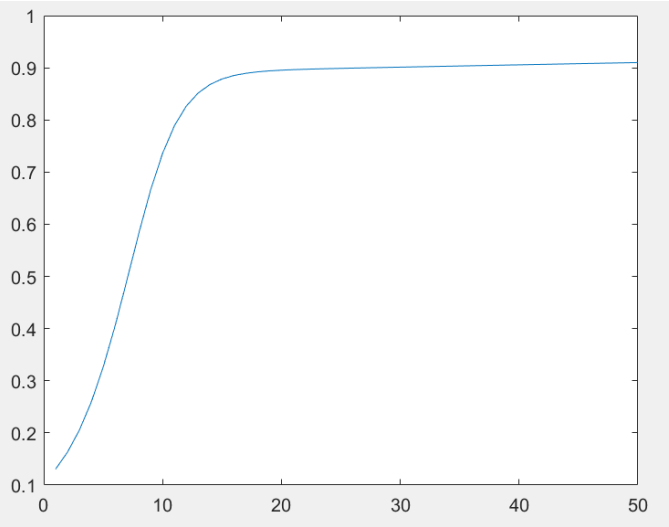

Fig. 3.28 One Example of Properly Executed Fidelity versus no. of Iteration Plot (m=2, Training Noise: x-flip, Testing Noise: y-flip, probability of flipping = 10%)

GHZ State

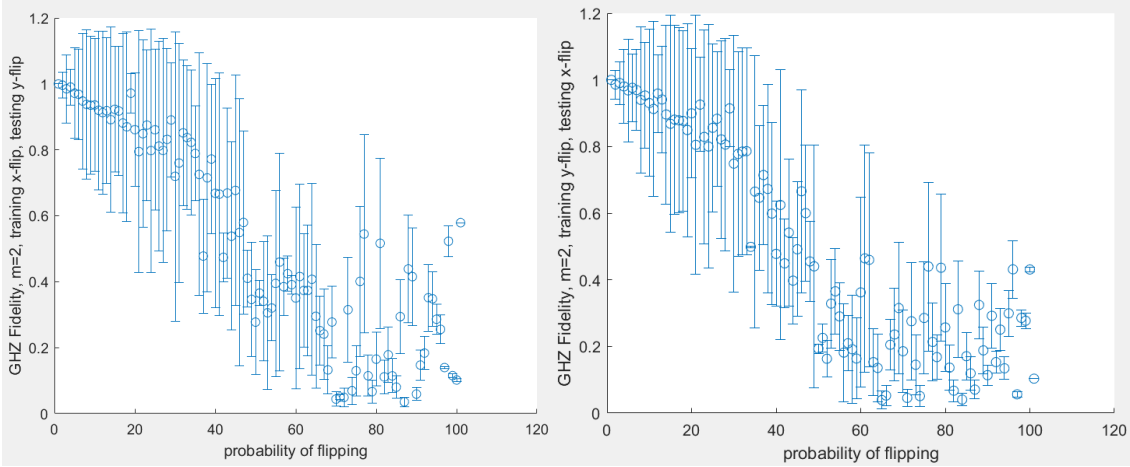

Fig. 3.29(a) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: y-flip)

Fig. 3.29(b) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: x-flip)

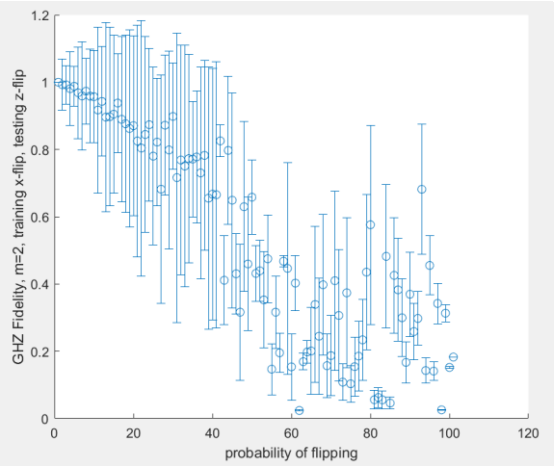

Fig. 3.29(c) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: z-flip)

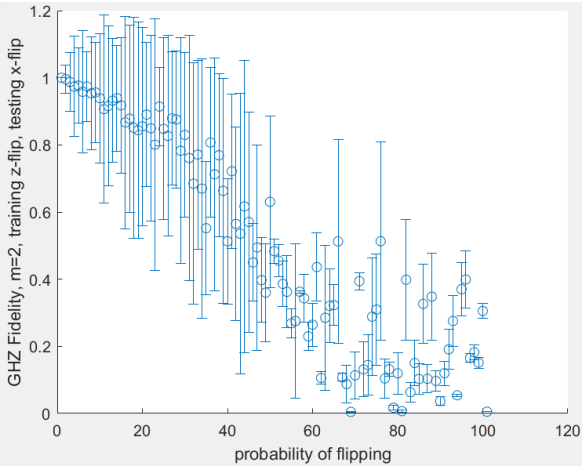

Fig. 3.29(d) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: x-flip)

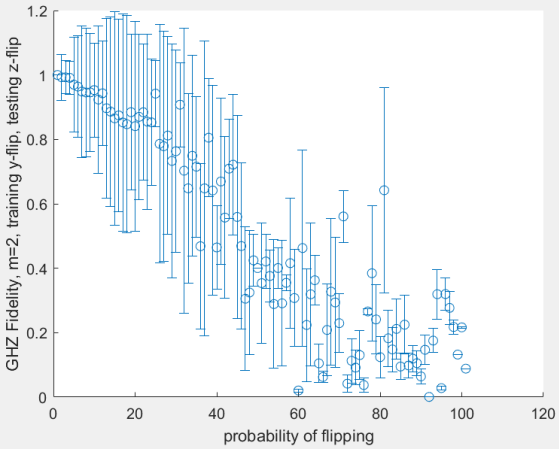

Fig. 3.29(e) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: z-flip)

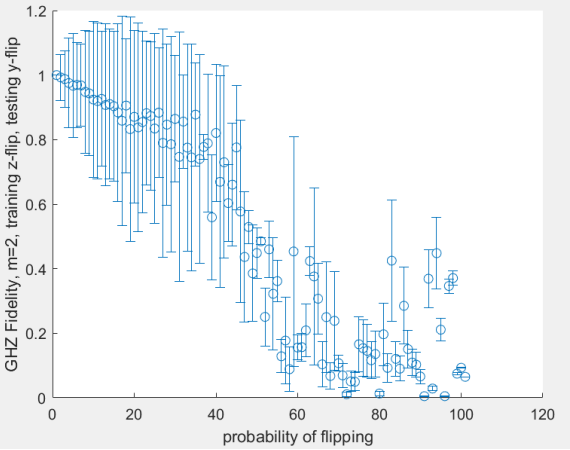

Fig. 3.29(f) GHZ-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: y-flip)

Compared to GHZ states for m=2 but with the same training and testing noise, the variances of the denosed states in this this case are significantly larger regardless of what types of training and testing noise it is (as long as they are different). Furthermore, the output fidelity almost decreased in the linear fashion. This suggests the QDAE is not functioning as that's how basically how the input fidelity decreased. It is not able to tell what are the noises and what are the state of interest from the testing data.

W State

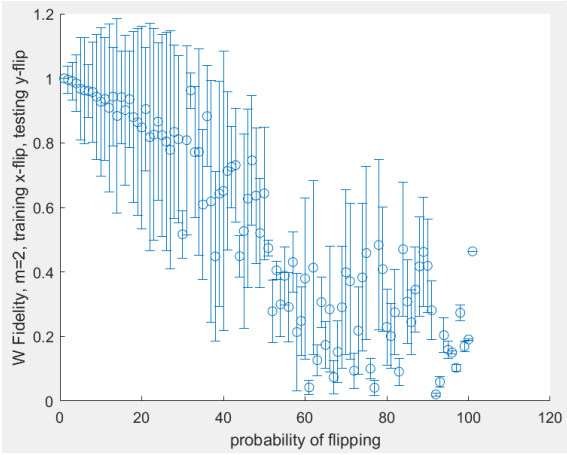

Fig. 3.30(a) W-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: y-flip)

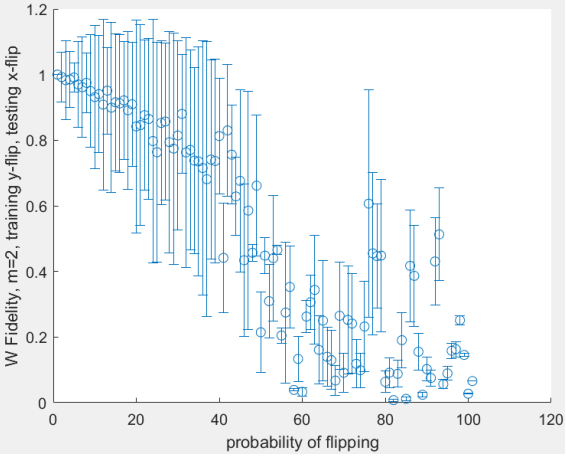

Fig. 3.30(b) W-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: x-flip)

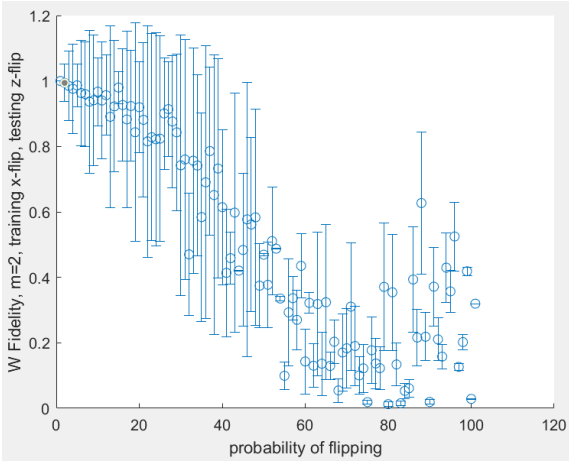

Fig. 3.30(c) W-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: z-flip)

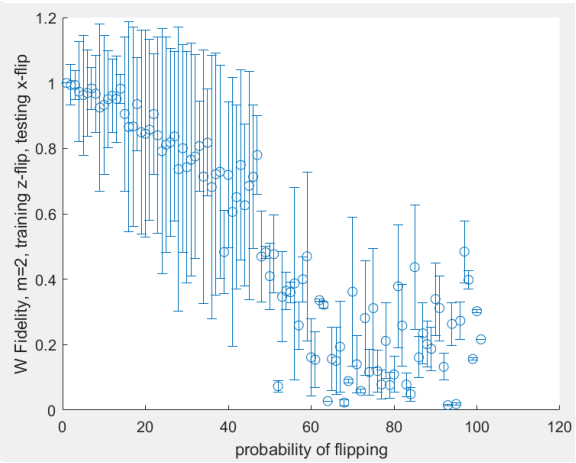

Fig. 3.30(d) W-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: x-flip)

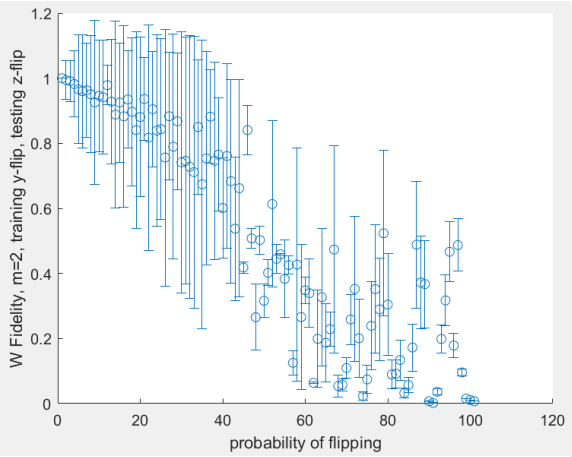

Fig. 3.30(e) W-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: z-flip)

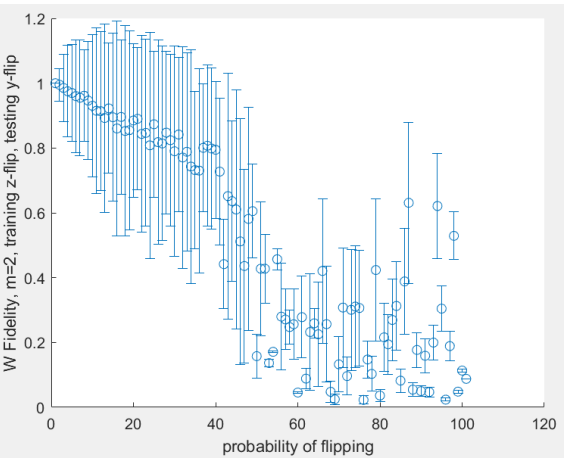

Fig. 3.30(f) W-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: y-flip)

The plots for W states have the similar patterns as the ones for GHZ states. The QDAE is not functioning.

Zero State

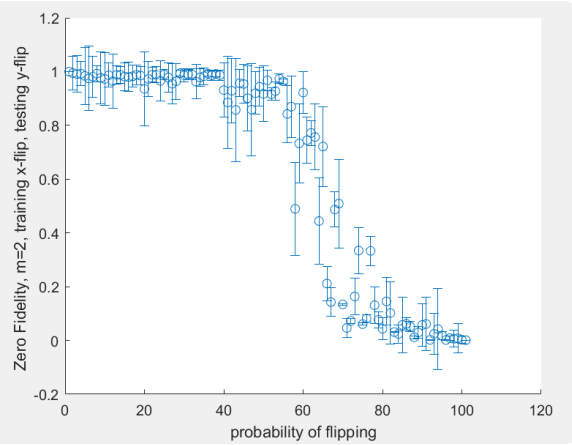

Fig. 3.31(a) Zero-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: y-flip)

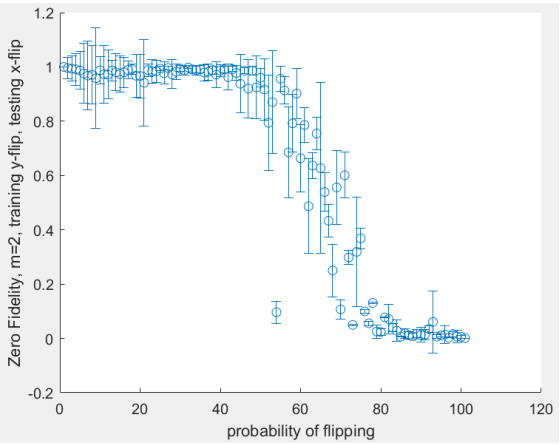

Fig. 3.31(b) Zero-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: x-flip)

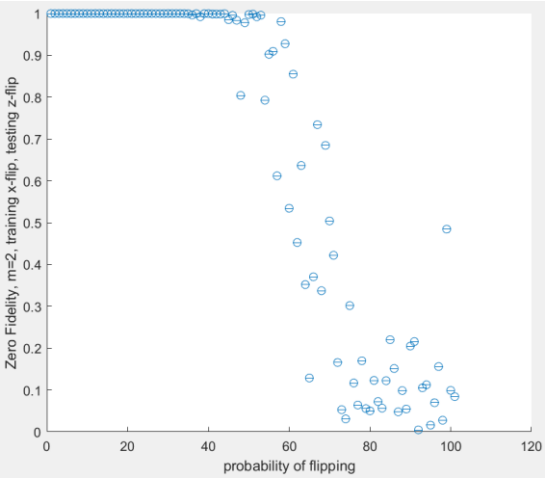

Fig. 3.31(c) Zero-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: z-flip)

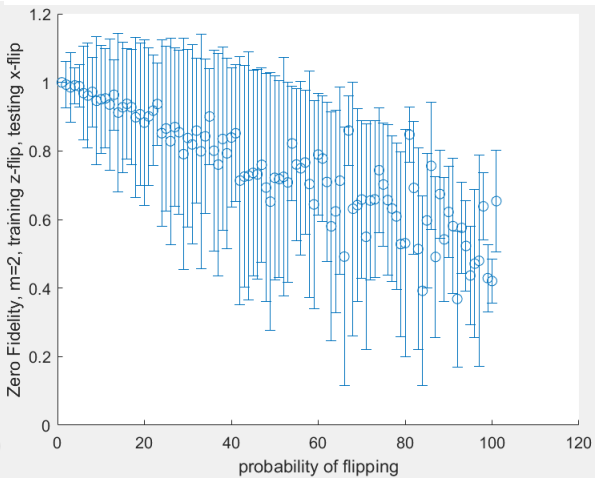

Fig. 3.31(d) Zero-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: x-flip)

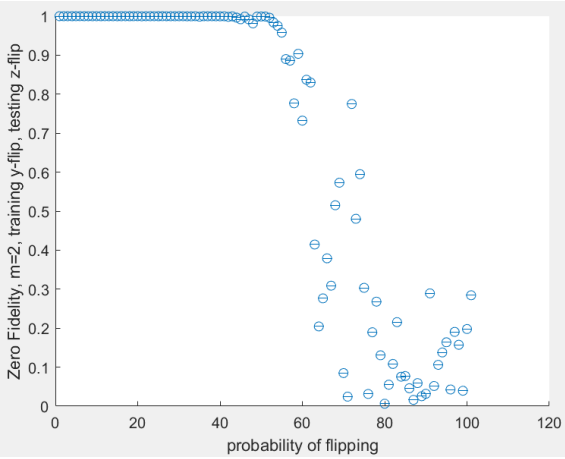

Fig. 3.31(e) Zero-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: z-flip)

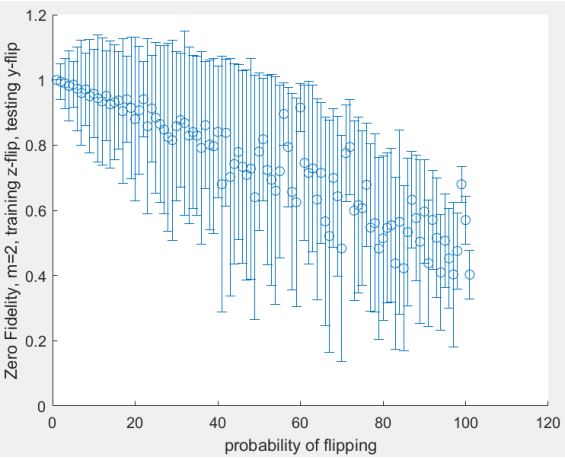

Fig. 3.31(f) Zero-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: y-flip)

There are something interesting in the plots for the Zero states. When the training and testing noises are x-y-flip pair, regardless of which one is training noise and testing noise, 3 segments can be seen clearly, just like the same traning and testing noise case:

1. the functioning interval (approximately from probability 0 to between 50% and 60%), in which the output fidelitie are almost 1 but with certain degree of variances;

2. the transitional period (immediately after the functioning interval until around 80%), in which the output fidelities dropped evenly;
3. the non-functioning interval (immediately after the transitional period), in which the output fidelities are basically fluctuating aound 0.

This pattern is almost like the Zero states with the same training and testing noise case, except the variance is slightly larger. This suggests that in these 2 cases the QDAE can actually do a decent denoising job.

When the QDAE is trained on x,y-flip noise and tested on z-flip noise, two segments can be observed on the plots:

1. the functioning interval (approximately from probability 0 to between 50% and 60%), in which the output fidelitie are almost 1 and practially without variances;
2. the transitional period (immediately after the functioning interval), in which the output fidelities dropped in the overall trend;

There is no well-defined non-functioning interval. In the functioning interval the denoising performance is exceptional well. If we account for the fact that z-flip will not alter Zero states, it means that the testing data are all noiseless. Therefore, at the functioning interval, it is reasonable that the denoising is perfect, as long the the QDAE knows what is that state of interest. However, that fact that there are the transitional period suggests that if the training data is corruputed severely enough, QDAE will not be able to recognize what is the original clear states even at the training stage.

Finally, when the QDAE is trained on z-flip noise, it means there is no noise present in the training data and naturally it cannot denoise because it has no concept of ‘noise’.

Plus state

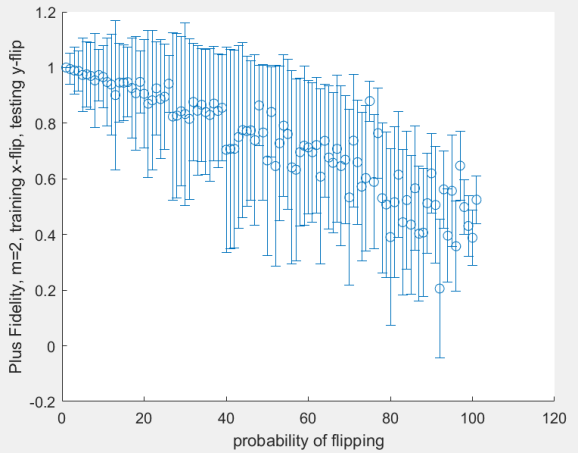

Fig. 3.32(a) Plus-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: y-flip)

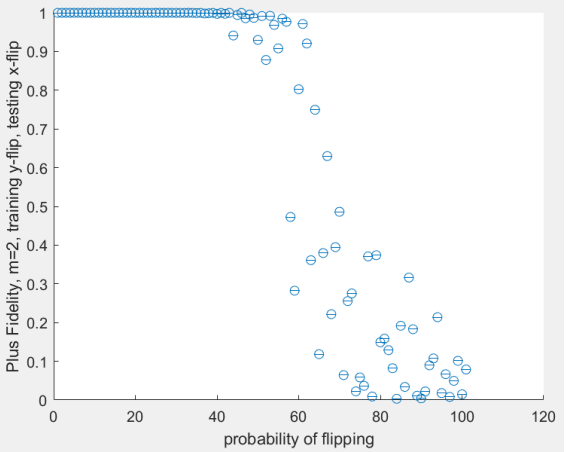

Fig. 3.32(b) Plus-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: x-flip)

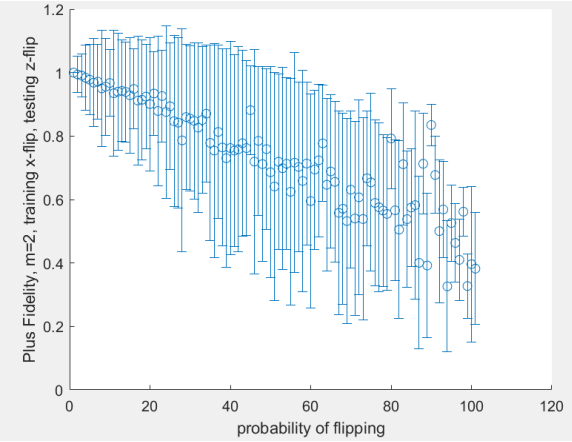

Fig. 3.32(c) Plus-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: x-flip, Testing Noise: z-flip)

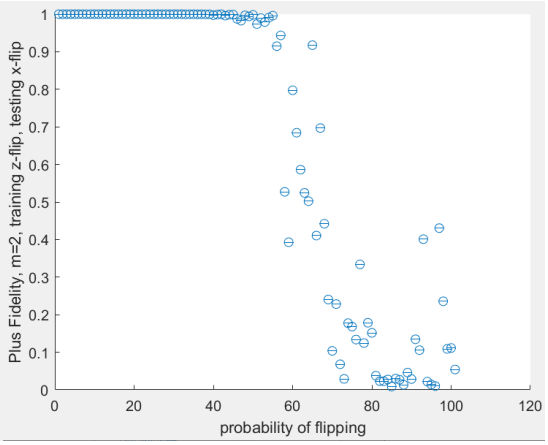

Fig. 3.32(d) Plus-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: x-flip)

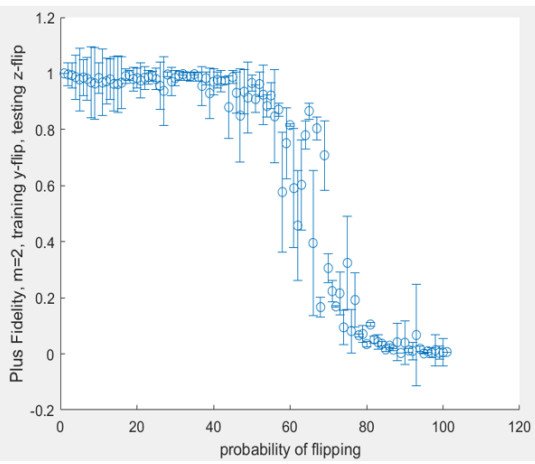

Fig. 3.32(e) Plus-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: y-flip, Testing Noise: z-flip)

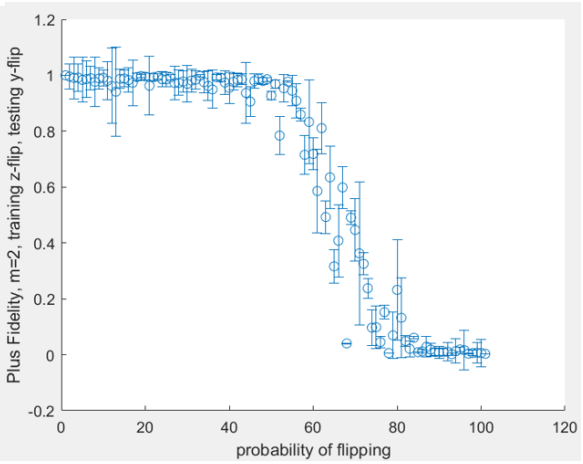

Fig. 3.32(f) Plus-State Fidelity versus Probability of Flipping Plot (m=2, Training Noise: z-flip, Testing Noise: y-flip)

For Plus states, the situation is very similar to the Zero states. When the training noise and testing noise pair is the one actually alters the states (in this case y-z-flip pair), the QDAE is functioning for the functioning interval (also approximately from probability 0 to between 50% and 60%), and 3 segments can be seen as well. When the testing noise is the one that leave the states unchanged (in this case x-flip), the QDAE recognized what is the state of interest (clean Plus state) within the same functioning interval and got confused when the noise is too large. When the QDAE is trained on the trivial noise (x-flip), it cannnot do the denosing for either y or z-flip since it has no idea what is noise due to the absence of noise in the training stage.

3.2.3 Different Training and Testing States with the Same Training and Testing Noise

Another interesting question to ask it what happens if the same training and testing noise and different training and testing state are used instead. GHZ-W pair, GHZ-Zero pair and W-Zero pair are tested for study with iterations per training set 50 to avoid over-

training. For each result, the loss-versus-iteration plot is checked to make sure the training was executed properly.

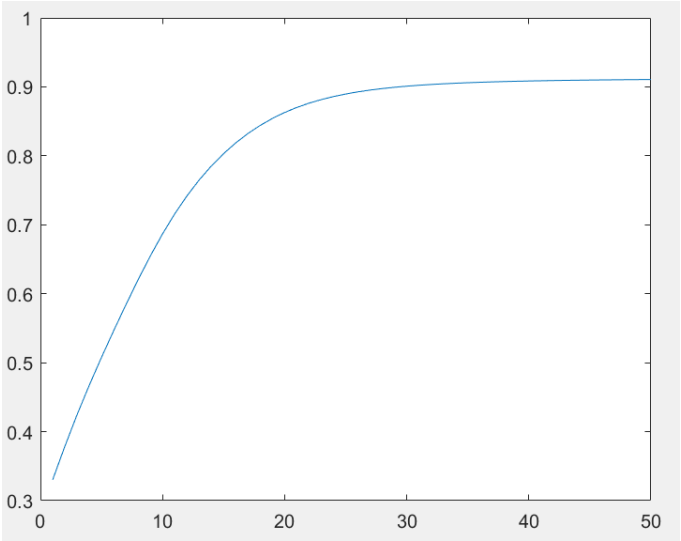

Fig. 3.33 One Example of Properly Executed Fidelity versus no. of Iteration Plot (x-flip, m=2, Training State: GHZ, Testing State: W, probability of flipping = 10%)

X-Flip

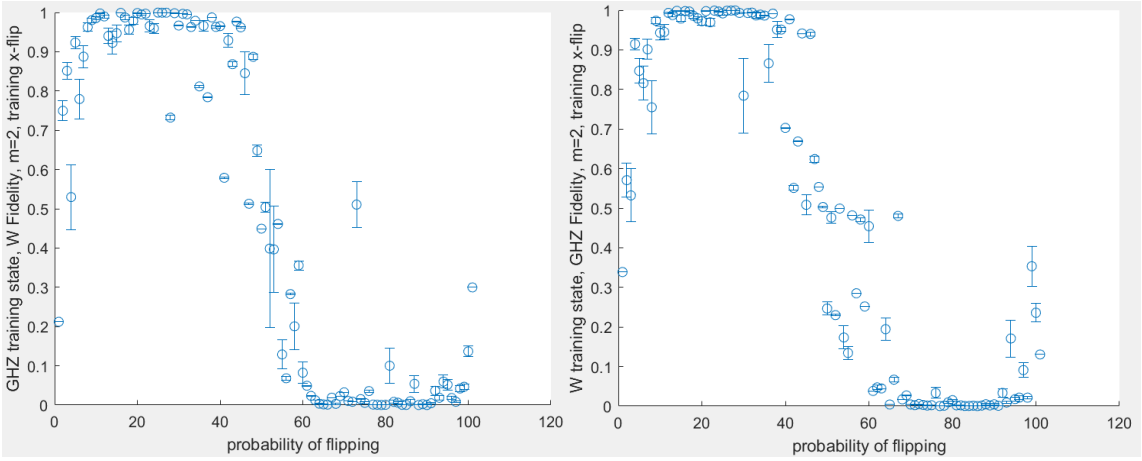

Fig. 3.34(a) Fidelity versus Probability of Flipping Plot x-flip (m=2, Training State: GHZ, Testing State: W)

Fig. 3.34(b) Fidelity versus Probability of Flipping Plot x-flip (m=2, Training State: W, Testing State: GHZ)

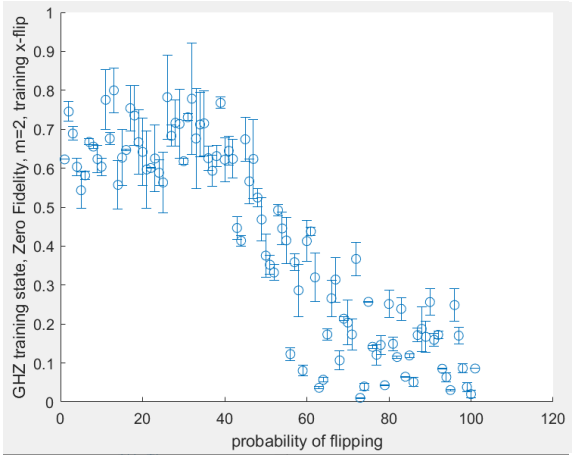

Fig. 3.34(c) Fidelity versus Probability of Flipping Plot x-flip (m=2, Training State: GHZ, Testing State: Zero)

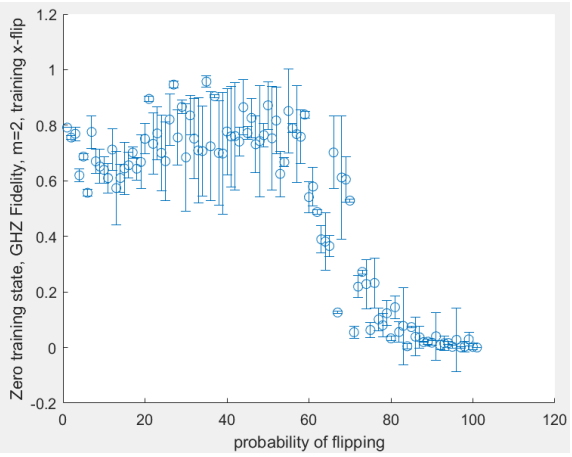

Fig. 3.34(d) Fidelity versus Probability of Flipping Plot x-flip (m=2, Training State: Zero, Testing State: GHZ)

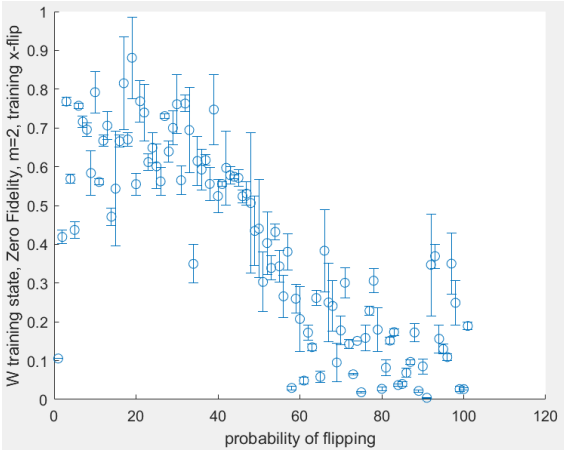

Fig. 3.34(e) Fidelity versus Probability of Flipping Plot x-flip (m=2, Training State: W, Testing State: Zero)

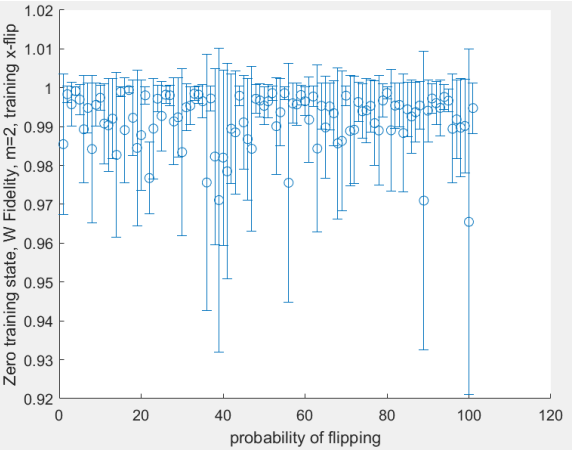

Fig. 3.34(f) Fidelity versus Probability of Flipping Plot x-flip (m=2, Training State: Zero, Testing State: W)

For GHZ-W pair, no matter which one is training or testing state, the plot shows the similar pattern:

1. When the fidelity is too high, the QDAE does not work;
2. the functioning interval (approximately from around probability 10% to between 40% and 50%), in which the output fidelitie are almost 1 with rather small variances;

3. the transitional period (immediately after the functioning interval until around 60%), in which the output fidelities has a overall dropping trend;
4. the non-functioning interval (immediately after the transitional period to right before 100%), in which the output fidelities are basically 0.
5. When the flipping probability is almost 100%. The output fedality increased for both cases.

Both plots have a skewed 'S' shape due to the aforementioned 5 segements. Compared to the same-state-different-noises case, the transitional period is shorter and steeper. For the lower output fidelities at the beginnig, it is possible that the DAE mistook the testing state as the noise due to there is no other state as noise. The suprising increase at the end is rather mysterious and an explanation is owed. Nevertheless, when the input fidelity is between 10% and 50% the QDAE can do the decent denoising job.

GHZ-Zero pair also have the similar pattern for both cases:

1. The initial output fidelity is low but keeps increasing with a rather low rate until when the probability of flipping is around 50%. The variances are moderately large;
2. The fidelity deceases from that point onwards. The variances are smaller compared to the first interval.

The denoising performance for GHZ-Zero pair is not as good as for GHZ-W pair. It is not suitable for practical use.

For W-Zero pair, when W is the training state and Zero is the testing state, the QDAE does not work, and the pattern of the fidelity-versus-probability-of- flipping plot resembles that of GHZ-Zero pair. When Zero is the training state and W is the testing state, a unique feature appeared in the plot: the variances are very different for different input fidelities but even the largest variance is considered to be small (0.0444). The mean of the testing fidelity is always above 0.96 regardless of in input fidelity. This pattern suggests

that Zero state is an excellent training agent for W state under x-flip noise, because somehow it helped to overcome the functional limit, so that there is no that limit anymore: the QDAE work for any input fidelity for deployment, regardless how high or how low the fidelity is for the states need to be denoised. Below are some of the basic statistical results:

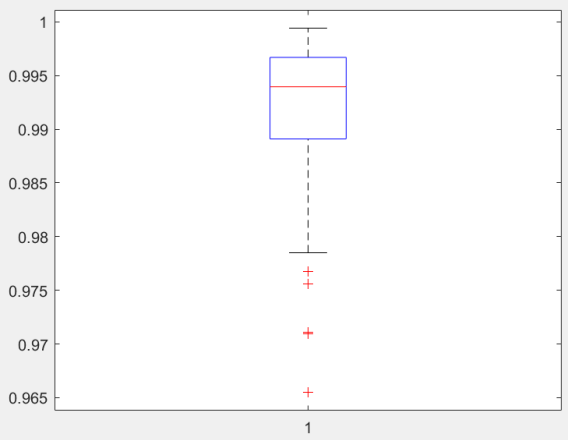

Fig. 3.35(a) Boxplot of the mean of output fidelities, x-flip (m=2, Training State: W, Testing State: Zero)

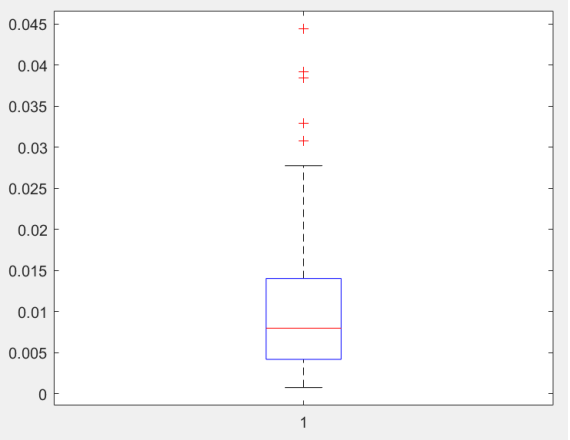

Fig. 3.35(b) Boxplot of the mean of output fidelities variances, x-flip (m=2, Training State: W, Testing State: Zero)

output fidelities(x-flip)	max	min	mean	median
means	0.9994	0.9655	0.9917	0.994
variance of means	0.0444	7.42E-04	0.0104	0.008

Table 3-1 Statistical results for the output fidelities (x-flip)

The box plots also show that the output fidelities are steadily high (near 1).

Y-Flip

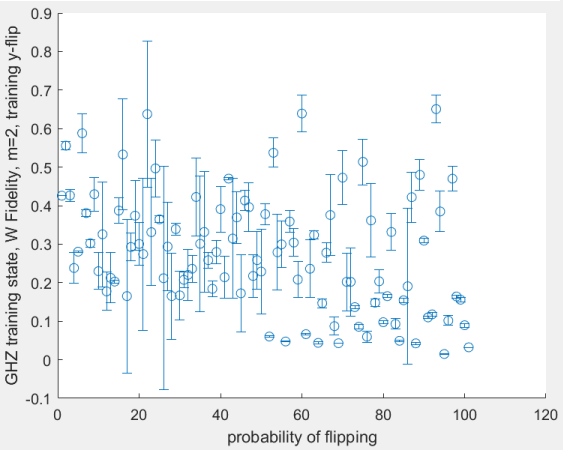

Fig. 3.36(a) Fidelity versus Probability of Flipping Plot y-flip (m=2, Training State: GHZ, Testing State: W)

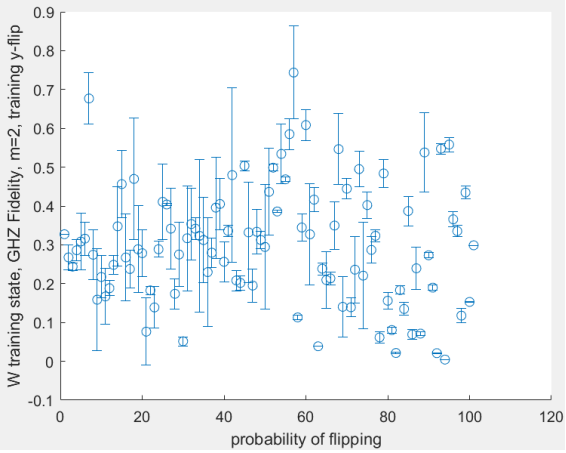

Fig. 3.36(b) Fidelity versus Probability of Flipping Plot y-flip (m=2, Training State: W, Testing State: GHZ)

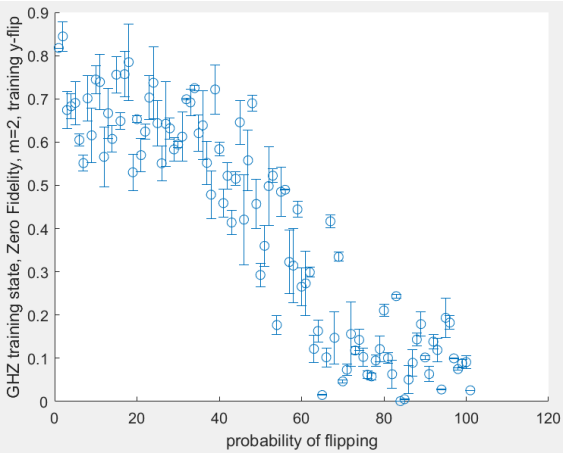

Fig. 3.36(c) Fidelity versus Probability of Flipping Plot y-flip (m=2, Training State: GHZ, Testing State: Zero)

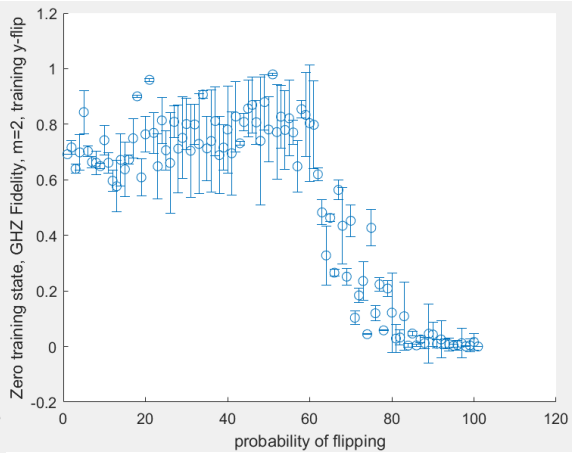

Fig. 3.36(d) Fidelity versus Probability of Flipping Plot y-flip (m=2, Training state: Zero, Testing State: GHZ)

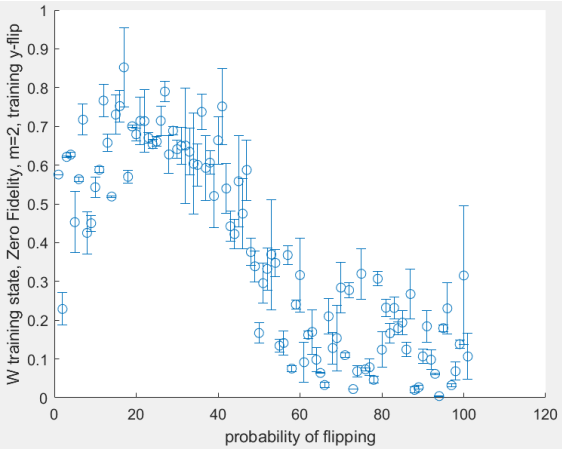

Fig. 3.36(e) Fidelity versus Probability of Flipping Plot y-flip (m=2, Training State: W, Testing State: Zero)

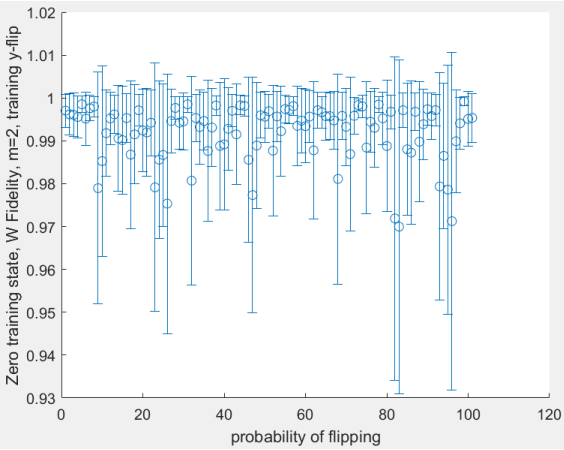

Fig. 3.36(f) Fidelity versus Probability of Flipping Plot y-flip (m=2, Training State: Zero, Testing State: W)

For GHZ-W pair, it is little pattern to talk about, the whole plot for both cases are almost entirely random, there is no correlation to be found between the input and output fidelities. The only commonality shared by both plots is that the variances (error bars) becomes smaller for lower input fidelities but it is hard to quantify the transition point/interval. In any case, QDAE cannot do denoising for y-flip.

For GHZ-Zero pair, when the training state is GHZ and testing state is Zero, the DAE is not functioning as the output fidelity dropped basically in the same manner as the input fidelity, except with more fluctuations. when the training state is Zero and testing state is GHZ, it follows the similar pattern as for x-flip case, but still, can be used for practical purpose.

For W-Zero pair, the plots are of the same pattern as x-flip case: when W is the training state and Zero is the testing state, the QDAE does not work, and the pattern resembles that GHZ-Zero pair's case. When Zero is the training state and W is the testing state, the mean of the testing fidelity is always above 0.97 regardless of in input fidelity with different variances for different input fidelities but small in general (the largest is 0.0394). This

means that Zero state is still an excellent training agent for W state for y-flip noise, which helped to remove the functional limit. Below are some of the basic statistical results:

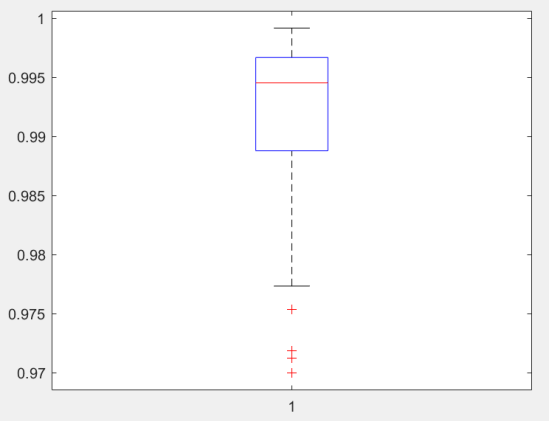

Fig. 3.37(a) Boxplot of the mean of output fidelities, y-flip (m=2, Training State: W, Testing State: Zero)

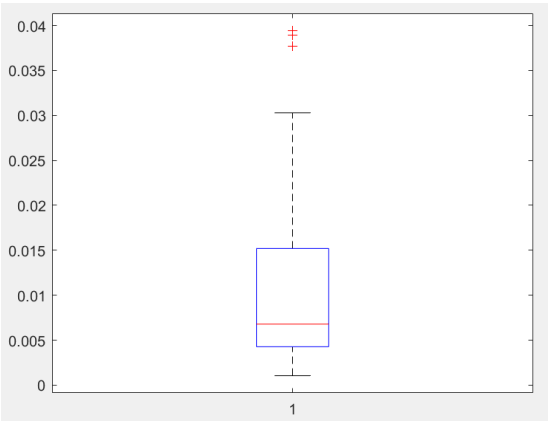

Fig. 3.37(b) Boxplot of the mean of output fidelities variances, y-flip (m=2, Training State: W, Testing State: Zero)

output fidelities(y-flip)	max	min	mean	median
means	0.9992	0.97	0.992	0.9945
variance of means	0.0394	0.0011	0.0104	0.0068

Table 3-2 Statistical results for the output fidelities (y-flip)

The boxplots show the stability and high quality of the output fidelities (near 1).

Z-Flip

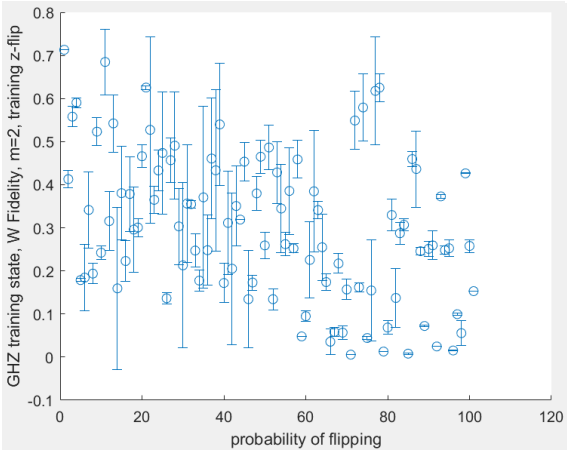

Fig. 3.38(a) Fidelity versus Probability of Flipping Plot z-flip (m=2, Training State: GHZ, Testing State: W)

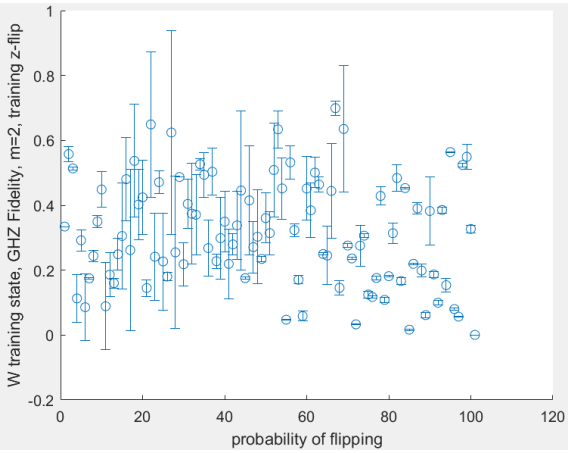

Fig. 3.38(b) Fidelity versus Probability of Flipping Plot z-flip (m=2, Training State: W, Testing State: GHZ)

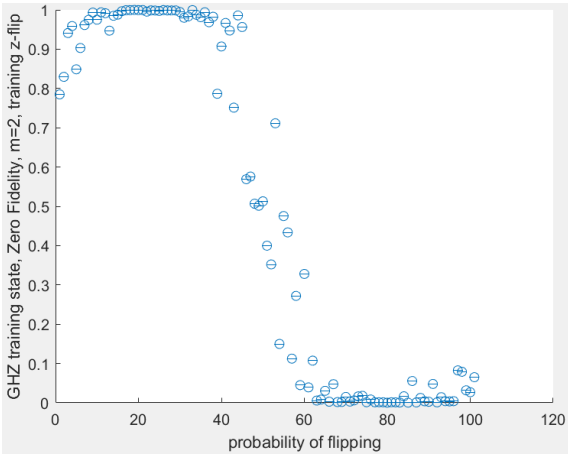

Fig. 3.38(c) Fidelity versus Probability of Flipping Plot z-flip (m=2, Training State: GHZ, Testing State: Zero)

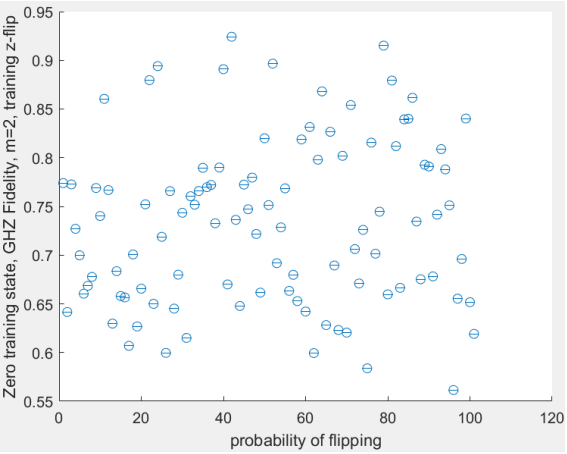

Fig. 3.38(d) Fidelity versus Probability of Flipping Plot z-flip (m=2, Training State: Zero, Testing State: GHZ)

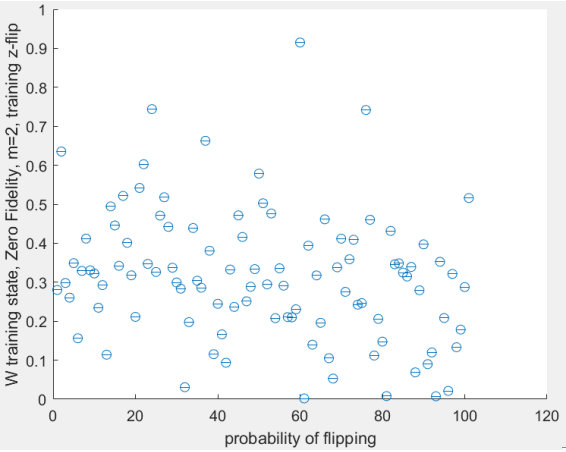

Fig. 3.38(e) Fidelity versus Probability of Flipping Plot z-flip (m=2, Training State: W, Testing State: Zero)

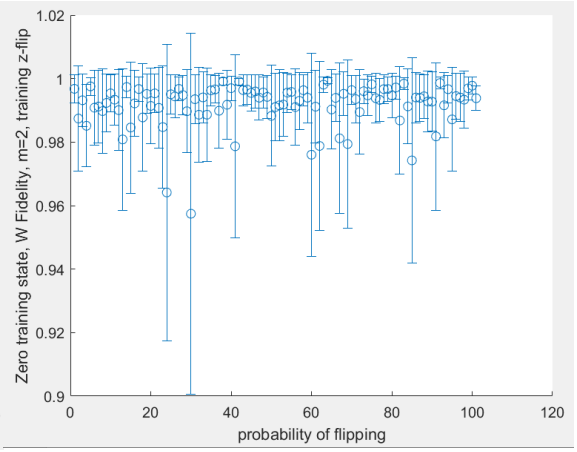

Fig. 3.38(f) Fidelity versus Probability of Flipping Plot z-flip (m=2, Training State: Zero, Testing State: W)

For GHZ-W pair, the plots are of the same kind as in the y-flip case. QDAE cannot be used.

For GHZ-Zero pair, when the training state is GHZ, QDAE works perfectly on Zero states when input fidelity is larger than around 0.5. Like the similar cases before, z-flip does not change zero state, hence the testing state is noiseless. But it was quite surprising the Zero states can be recognized as the state of interest, despite it was never seen in the training stage. When the training state is Zero, the output is entirely random in terms of input-output fidelity correlation but with no variance. This is quite different from the case in same training state with different training and testing noises (Zero states, training z-flip, testing x-flip), where the linear decrement in fidelity is roughly preserved. When the training is noiseless (z-flip on Zero state), but the testing states are of another type, the linear decrement is nowhere to be found. It is likely when the testing state was not seen during the straining stage and it is corrupted by noise, QDAE cannot tell which is the state of interest anymore. Comparatively, in the ‘Zero states, training z-flip, testing x-flip’ case, despite it does not know what is a ‘noise’, the state of interest it learned from training was still the same as the one in the testing stage, hence the different patterns in the respective plots.

For W-Zero pair, when W is the training state and Zero is the testing state, the output fidelity has no correlation with the input fidelity. This time however, it is not sure to the over-training issue since it is checked that the training was executed successfully:

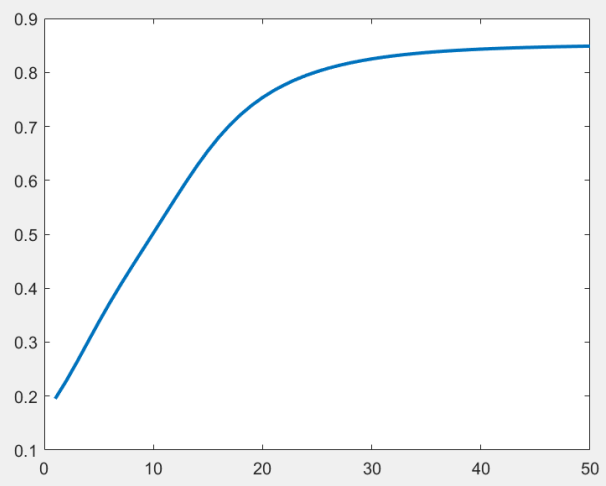

Fig. 3.39 One Sample of Properly Executed Fidelity versus no. of Iteration Plot (z-flip, m=2, Training State: W, Testing State: Zero, probability of flipping = 10%)

This suggests that the QDAE learned a denoising scheme from noisy W states that does not work on noisy Zero state.

When Zero is the training state and W is the testing state, like the cases for x and y-flip, functional limit vanished. Below are some of the basic statistical results:

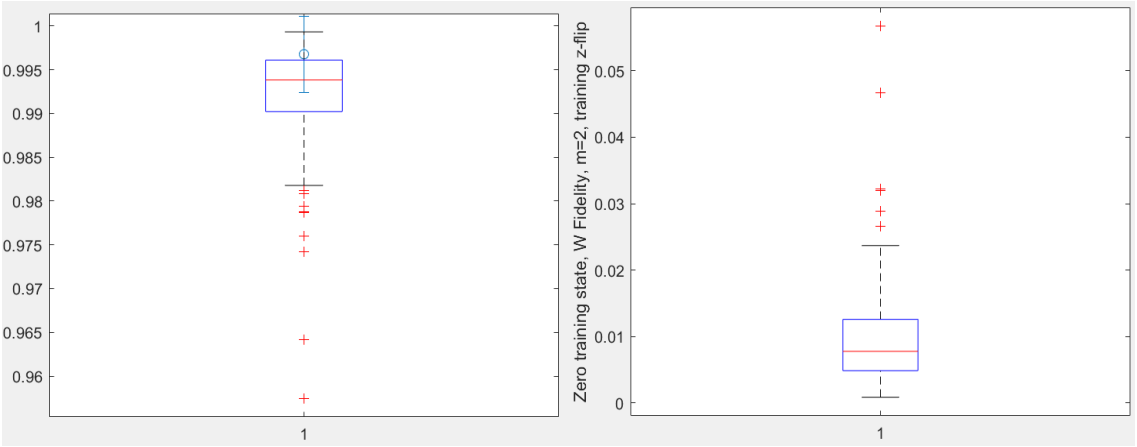

Fig. 3.40(a) Boxplot of the mean of output fidelities, z-flip (m=2, Training State: W, Testing State: Zero)

Fig. 3.40(b) Boxplot of the mean of output fidelities variances, z-flip (m=2, Training State: W, Testing State: Zero)

output fidelities(z-flip)	max	min	mean	median
means	0.9993	0.9574	0.9918	0.9938
variance of means	0.0568	8.65E-04	0.0104	0.0078

Table 3-3 Statistical results for the output fidelities (z-flip)

It can be seen that the output fidelities are quite stably stayed at a high range (near 1). Together with the previous results for x and y-flip, it can be concluded that training with Zero states can drastically improve the denoising performance of QDAE on W states.

Chapter 4

Discussions, Conclusions and Future Work

4.1 Discussions and Conclusions

4.1.1 Compressive Autoencoder

a) Experiment results analysis

As shown in the results, the proposed method is an effective compression protocol for a real-valued vectors (data). As long as the transformation that did the compression is known, the vectors (data) can always be recovered by apply the inverse of that transformation. And since the transformation is always unitary, there is always another transformation (its inverse transformation) that can undo the compression. Unfortunately, the method does not apply to the compression of vectors (data) with complex-values entries. The reason behind is that the structure of the complex vector space is not congruent to a real vector space and certain rotations (the one that is needed to perform lossless compression) does not exist. However, this does not eliminate the possibilities that there are still complex unitary transformations can perform specific types of compression, given that some restrictions are satisfied. This needs further investigation and study to find out.

The loss-versus-iteration plots looks promising and the histogram of the first entry shows the compression is quite effective. Norm mean is very close to 1 and norm variance is rather small. All that means our two constraints are satisfied to a considerable extent after tuning the hyperparameters such as the learning rate and Lagrange multiplier. As mentioned before, this numerical solution won't be as accurate as an analytical result, so the next step is to find the unitary matrix that is closest to the trained one through quantum mechanical process. However, it is possible that the physical law will assist this step since it imposes the unitarity constraint.

b) In view of information entropy

The compression problem can be analyzed in terms of its entropy. The point is to compute the information entropy of our initial and final data. Von Neumann entropy is a generalization of Shannon entropy for quantum states. Shannon entropy of the associated with probability distribution of random variable X is defined as

$$H(X) \equiv H(p_1, p_2, .., p_n) \equiv -\sum_x p_x \, log_2 p_x$$

where p_x is the probability of certain outcome for X, while Von Neumann entropy is defined as

$$S(\rho) \equiv -\mathrm{tr}(\rho \ log \ \rho) = -\sum_x \lambda_x \, log_2 \lambda_x$$

where ρ is the density matrix of the quantum system. Conjugate transpose is just the ordinary matrix transpose in this case of real vectors. λ_x are the eigenvalues of ρ.

In the 3-dimensional complex vector case (with constraint),

$$\rho \equiv \sum_i p_i \, |\psi_i\rangle\langle\psi_i| = \frac{1}{3}\begin{bmatrix}1\\0\\0\end{bmatrix}[1 \quad 0 \quad 0] + \frac{1}{3}\begin{bmatrix}0\\1\\0\end{bmatrix}[0 \quad 1 \quad 0] + \frac{1}{3}\begin{bmatrix}0\\0\\1\end{bmatrix}[0 \quad 0 \quad 1] = \frac{1}{3}\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

where the bases are not canonical. The Von Neumann entropy of the uniformly distributed collection of input vectors input is

$$S(\rho) \equiv S(\rho) = -\sum_{x} \lambda_x \, log_2 \lambda_x = -3\left(\frac{1}{3} log \frac{1}{3}\right) \approx 1.585$$

Since the output could be fully described by three canonical axes, the Von Neumann entropy would be the same (this point will be explained more in detail in the final report).

c) Potential problems identified and resolved

From Shannon's source coding theorem, in principle such a compression process could be lossless. However, since the unitary operation found is rather an approximation since gradient descent is a numerical method, there will be inaccuracy, however negligible. It takes the physical process of the quantum device to correct this discrepancy (the law of quantum mechanics imposes the transformation to be unitary).

The norm of each entry of the initialized random unit vectors for training weren't generated uniformly (e.g. Fig.) despite fulfilling the normalization condition. Initially this might appear as a problem since samples are better drawn from randomized (uniform) distribution. The underlying reason of such an apparent non-uniform distributed vector is due to the constraint imposed. Imagine for 3-dimensional real space case, if the vectors are all initialized on x-y plane, then v3 (the green bar in Fig. 4(a)) is going to be only non-zero at value zero, because there is no variation of z value in x-y plane. Therefore, this is nothing but a phenomenon consequential to the formulation of the problem and ought to be expected.

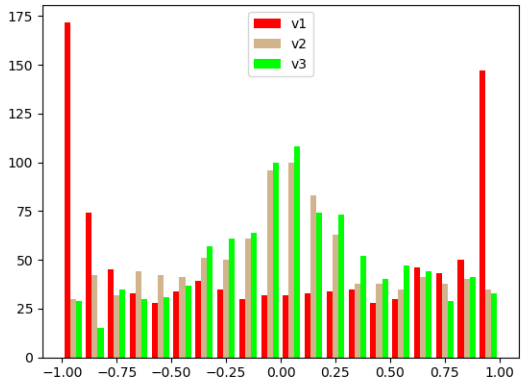

Fig. 4.1(a) Histogram of real input vectors (1000 samples)

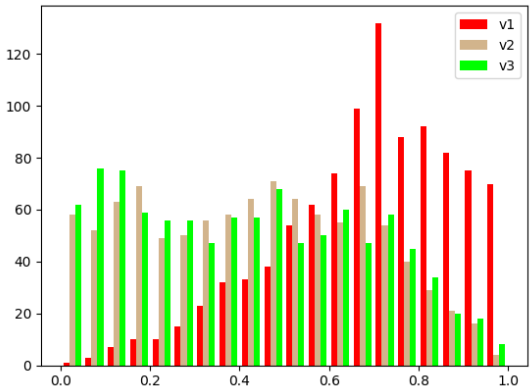

Fig. 4.2(b) Histogram of the norms of complex input vectors (1000 samples)

d) Changes made compared to the original approach

Originally, as proposed by one of the papers which we have been following closely [1], the plan was to use a well-studied neural network structure, namely autoencoder, to compress the state vectors and re-construct them to compute the loss. However, it turned out that the compression process must be a unitary process the at the middle part (Fig. 2.2(b)) and the number of nodes cannot be reduced as the structure of a convectional auto-encoder. Moreover, we proved that as long as the constraints mentioned in Section 3.1 are satisfied, the output vector will be the one of our interest. Decompress can be done the simply by applying the inverse of the trained unitary matrix to the output vector, since unitary transformation is fully reversible. Therefore, there is no need to implement the decoder part. Our final neural network structure is simply a perceptron model, consists of one input layer and one output layer without hidden layer, which is capable of handling our defined task.

Secondly, Pepper et al. used a set of optical apparatus to implement their autoencoder and used physical process (rotation of waveplate) to execute their optimization, without invoking real neural network architecture. In our case, an explicit artificial neural network structure (technically a two-layer perceptron model with activation function set to 1) is

designed and optimization technique is applied onto the network. According to Zeilinger et al.'s work [19], it is always possible to use a set of optical devices to realize the discovered unitary operator for practical use. Furthermore, it is likely that the devices will automatically correct the inaccuracy introduced by constrained optimization because due to the law of physics (any coherence quantum process is unitary).

e) Compressibility and generalizability

The compressibility of the lossless compression totally depends on the inherent data structure. If there exists a way to reparametrize the input data with another set of base which only occupy a subspace of the original vectors space in which the original input vectors are defined, then it is possible to do lossless compression by reparameterization. The smaller the number of new bases can be used to fully describe the original data, the more the data can be compressed.

In terms of the generalizability, although we only showed 3-dimensional-to-2-dimensional compression, the method can be generalized to arbitrary n_1-dimensional-to-n_2--dimensional compression, provided given the currently formulated loss function and the data vectors to be compressed span only a subspace n_2 of the n_1-dimensional space. As a rule of thumb, the smaller of the subspace dimensionality the vectors span, the more they can be compressed. However, one foreseeable issue is that, as the dimensional gets higher, the loss function will become much complicated and much demanding to the computational resources.

In a nutshell, the proposed method is a valid alternative to the approach given by Pepper et al. Both methods could potentially help to make quantum communication more efficient and less costly.

4.1.2 Quantum Denoising Autoencoder

From all the experiments conducted above, there is a general gauge for how good the generalizability of the proposed QADE is:

a) For the same training state/noise and testing state/noise

Other than the GHZ states in the original work[2], we have shown that the QDAE with the architecture $[m, 1, m]$ (m input nodes, 1 hidden layer with 1 node, m output nodes) is able to denoise the quantum states W state, Zero state and Plus state corrupted by x, y and z-flip noise. This suggests a wider range of application for QDAE. However, the quality of the denoising performance depends on the value of m. The larger the m (larger qudits), the worse the quality in general regardless of which quantum state is using. When m comes to 5, the quality is too unstable to use. Since the training time of QDAE increases exponentially with respect to m on the classical computer, the runtime becomes too large to test the cases with even larger m. Therefore, it becomes unrealistic to test those cases within the span of FYP's timeline.

b) For the same training and testing state with different training and testing noise

DQAE has different denoising effects on different states. It completely has no effect on GHZ states and W states, but it works non-trivially on Zero states and Plus states with non-trivial flip noise, that is to say, x and y-flip for Zero states, y and z-flip for Plus states, regardless of which flip is training noise and which flip is testing noise. The commonality of Zero and Plus states is that both of them have a trivial flip (z-flip for Zero states and x-flip for Plus states). The test was only done for $m = 2$ case due to the time constraint, but the this is also expected to have the most representative result due to its quality of denoising in the previous experiment.

c) For different training and testing states with the same training and testing noise

QDAE only works for certain types of noises with certain training state and testing state at the same time. Those cases include:

1. x-flip with GHZ states as training state and W states as testing state;
2. x-flip with W states as training state and GHZ states as testing state;
3. x-flip with GHZ states as training state and Zero states as testing state (trivial case).
4. Zero states as training state and W states as testing state.

The best denoising results occurred when the training states are Zero states and testing states are W states. In this case, regardless of the type of flip noise, the output fidelities are constantly high (near 1) regardless of the input fidelity. It is due to the vanish of the functional limit appeared in all other working cases. Zero states seem to have excellent functionality as the training agent for denoising W states.

The test was done exhaustively for the three selected quantum states. Like the previous experiment, only $m = 2$ case was conducted for the same reason.

d) Tabulated summaries of the testing results

Training, Testing States	Training, Testing Noises	Denoising Quality
Same	Same	Roughly same for all states, decreases as m increases
Same	Different	Good on Zero, Plus, x-flip (training/testing), y-flip(testing/training), m=2, other cases no effect
Different	Same	Works for x-flip, GHZ (training/testing), W(testing/training), m=2; x-flip GHZ (training), Zero(testing) (trivial case). Excellent for Zero(training), W(testing), all flips; other cases no effect

Table 4-1 Generic Summary of the Denoising Qualities

GHZ State, m=2		Testing Noise		
		x-flip	y-flip	z-flip
Training Noise	x-flip	Good	Not working	Not working
	y-flip	Not working	Good	Not working
	z-flip	Not working	Not working	Good

Table 4-2 Summary of the Denoising Qualities (GHZ states)

W State, m=2		Testing Noise		
		x-flip	y-flip	z-flip
Training Noise	x-flip	Good	Not working	Not working
	y-flip	Not working	Good	Not working
	z-flip	Not working	Not working	Good

Table 4-3 Summary of the Denoising Qualities (W states)

Zero State, m=2		Testing Noise		
		x-flip	y-flip	z-flip
Training Noise	x-flip	Good	Working	Trivial
	y-flip	Working	Good	Trivial
	z-flip	Trivial	Trivial	Trivial

Table 4-4 Summary of the Denoising Qualities (Zero states)

Plus State, m=2		Testing Noise		
		x-flip	y-flip	z-flip
Training Noise	x-flip	Good	Trivial	Trivial
	y-flip	Trivial	Good	Working
	z-flip	Trivial	Working	Trivial

Table 4-5 Summary of the Denoising Qualities (Plus states)

x-flip, m=2		Testing State		
		GHZ	W	Zero
Training State	GHZ	Good	Working	Not working
	W	Working	Good	Not working
	Zero	Not working	Excellent (no functional limit)	Good

Table 4-6 Summary of the Denoising Qualities (x-flip)

y-flip, m=2		Testing State		
		GHZ	W	Zero
Training State	GHZ	Good	Not working	Not working
	W	Not working	Good	Not working
	Zero	Not working	Excellent (no functional limit)	Good

Table 4-7 Summary of the Denoising Qualities (y-flip)

z-flip, m=2		Testing State		
		GHZ	W	Zero
Training State	GHZ	Good	Not working	Trivial
	W	Not working	Good	Not working
	Zero	Trivial	Excellent (no functional limit)	Good

Table 4-8 Summary of the Denoising Qualities (z-flip)

In a nutshell, QDAE showed certain level of generalizability under specific conditions. Relating to classically machine learning, this is comparable to the transfer learning technique where unseen samples can be processed successfully by the machine learning model (be it for classification, regression, prediction task etc.). This suggests more potential applications in quantum information processing and quantum machine learning.

4.2 Recommendation in Future Work

4.2.1 Compressive Autoencoder

a) Modification of loss function

As discussed in Section 3.1, the modification of the loss function to adapt lossy compression, or in other words, projecting data which inherently distributed in a higher vector space onto its subspace according to the degree of variance of the distribution of data along some appropriate direction is highly non-trivial. If this could be done, every task originally for principal component analysis can now be solved by constrained optimization. This is a topic worth further investigation.

b) Mixed state system

So far, only pure state quantum system has been studies in this thesis. The classical probability in the density matrix arose from the fact that different pure states were chosen to transmit different message in practical setting. However, the same method presumably could be generalized into for mixed quantum states as well. For future work, it is beneficial to investigate how to adapt the constrained optimization technique that have been using so far to compress mixed states.

4.2.2 Quantum Denoising Autoencoder

Further investigation into the generalization ability of QDAE:

- As mentioned in the conclusion, the QDAE with $[m, 1, m]$ architecture is not robust enough to denoise qudits for d larger than 4. It is possible that the QDAE with other architectures can raise this benchmark. For example, $[m, 3, 2, 1, 2, 3, m]$ could be tested. However, with more complicated structure, much more running time for training is expected.

- So far, the experiments are only done on for types of quantum states: GHZ, W Zero and Plus. More types of quantum states could be tested as well in order to discover more patterns and rules.

- So far, the experiments only covered flip noise. However, quantum states can be corrupted by other kinds of noise. For example, the phase noise can appear to change the relative phase of a pure state:

$$|\psi\rangle = x_0|0\rangle + x_1|1\rangle + x_2|2\rangle + \cdots x_n|n\rangle \ldots\ldots original\ \ state$$

$$|\psi'\rangle = x_0|0\rangle + x_1 e^{i\theta_1}|1\rangle + x_2 e^{i\theta_2}|2\rangle + \cdots x_n e^{i\theta_n}|n\rangle \ldots\ldots state\ \ corrupted\ \ by\ \ phase\ \ noise$$

QDAE's denoising ability can be tested against other types of noise such as this in the future work.

- For different train and testing noises with same states case, the testing on the four quantum states of interest is exhaustive. However, other types of quantum states could be tested in the future work to explore more hidden patterns and rules.

- Test on different training and testing state with the same noise type more case is exhaustive for the three selected states. In the future work, this testing can be done exhaustively and include many more other states (exponential increase of the totally number of combinations is expected). The reason behind why noisy Zero states as training agent can give excellent denoising results for noisy W state and the peculiar shape of the GHZ-W pair plots need further investigation as well.

- The most challenging generalizability test is on different training and testing state as well as different training and testing noises. This would require QDAE to have high level of abstraction for the pattern it learned. More sophisticated experiments are needed to be designed for this test.

Reflection on Learning Outcome Attainment

Throughout the learning journey of doing the final-year project, I have gained numerous valuable knowledge, skillsets as well as insights.

To begin with, I have learnt so much of the frontier knowledge. I have always been fascinated by the latest researches on quantum information and artificial intelligence. Through the final-year project, I had the opportunity of diving deeper into those topics as well as equipping myself with solid technical knowledge in the areas. These are extremely valuable assets as the foundation should I decided to pursue further study or even research career in the fields.

Moreover, I am much more capable of coding now. The first part of the project trained me to code in Python whereas the second part of the project trained me to code in MATLAB environment. Loads of coding tasks forced me to learn to translate what I learned and understood in mathematics (constrained optimization) as well as physics (quantum states manipulation) into the language of machine for the computation. In this way, those theoretical knowledges are no longer merely theoretical to me, but rather are the tools I can make use of to solve practical problems. From my point of view, this is the essence of being a successful engineer: to be able to adapt theory into the real-world scenario and solve practical problems.

Meanwhile, I also picked up various tools and learned how to use them: using Theano to solve a constrained optimization task; deploying runtime MATLAB in cross-platform environment; familiarizing with Linux operating system and its command line paradigm, etc. Finally, I learned to use build docker image and submit it to NSCC supercomputing resource to solve computational tasks which are too demanding for my personal computer. While acquired and honed these tools partly from the Internet, I also reached out to my

peers, Ph.D. students and professionals who are equipped with relevant skillsets for help in the time of need. Through this process, I realized the importance of consulting experts and adopting different tools for problem solving whenever they appear to be useful. This is true no matter in industry or academia.

Suffice it to say, through the final-year project, I have analyzed numerous data. Along the way I learned how to represent those data in the most understandable way, through visualization of plots and tables. Only by these means I was able to discover the patterns and the rules hidden beneath and possibly came out with valuable insights. Data analytics is not only a necessary ability for doing research, but also a trending and highly sought-after skill in the industry nowadays. Therefore, I believe what I learned from this project will be beneficial for my future employment.

In addition, I have learned to think out of the box when it comes to analyzing problems. For instance, I have never realized the connection between two remote topics: constrained optimization and principal component analysis. After hinted by Professor Jiang Xudong, I was able to soon discover the underlying similarities between those two topics in the context of data compression. This skill is important for working in industry as well as academia. A lot of times, the solution towards a certain problem can come from a seemingly unrelated area, so the ability to discover even the slightest trace of connection between topics or knowledges from different realms are critical. As another example, instead of following the method proposed by Pepper et al. to solve the quantum data compression task, I came out with another viable approach. In doing research, it is important to keep actively exploring all the possible ways of solving a problem and not let yourself be limited by the thinking of others.

I firmly believe that the research in quantum information and artificial intelligence will have both near-term and long-term impact to our society. As a matter of fact, we can already see industry has begun to adapt various technology developed from artificial intelligence research into their practice and boosted their productivities as well as the

economy [20]. On the other hand, despite having a certain amount of time from being fully matured, quantum technology is already on the highway of advancement. Leaders from different industry sectors are investing heavily in developing state-of-the-art quantum computing devices and quantum-technology-based start-ups are burgeoning all around the world [21]. It will not be a very long time when we can see how it can help to transform our society, from discovering new medicines to providing insurmountable communication security and many more to come.

Doing the final-year project also taught me to value and respect other people's intellectual properties more. I spent days and nights trying to figure out how to solve the problems and came to know the toils needed in exchange for the moments of enlightenment and discovery. To come out with something new is never easy. This is real for anyone. After I experienced this hardship, I came to know this truth by heart. Therefore, I also cherish the intellectual contributions made by other people. Amongst many things, I kept my integrity when it comes to the issue of plagiarism by never taking other's intellectual work for granted without acknowledgement. When citing the research results from other researchers, I tried to give proper and detailed references as much as I could. This is one of the most basic yet the most important ethics one must have to embark on a research career.

In some way, I was working in a cross-disciplinary environment. My project topic is multi-disciplinary in nature, standing at the intersection of physics, mathematics and engineering. My supervisor and co-supervisor own the expertise from different fields: one emphasizes more on engineering point of view while another often thinks from a theoretical physics perspective. My meetings with them were always one-to-one, meaning that they never had the chance to speak to each other about my project. It is only natural for me to feel challenging when I tried to deliver my discussion results with one professor to another, since they tend to use different technical languages and tackle the problem from different angles. Nevertheless, I persevered and managed to do just that. And through this process, I learned how to think the same problem from different perspectives and

fluently formulate and translate it from one to another. This further developed my versatility in terms of critical thinking and problem-solving skills.

Last but not least, I learned to do research independently. As much as my supervisor and co-supervisor have helped and guided me, I have to do independent study such as literature review and designing experiments. At times I stumbled upon unexpected results due to the bugs in the code or the inadequacy in the problem formulation. When situation came, I need to find out what exactly the problems are. This really trained my patience and the ability to troubleshoot independently. As the result, when I discussed with my supervisor and co-supervisor in the meeting, we could discuss based on my improved results as well as new understanding. This is much productive than asking for help to find elementary mistakes and debugging from scratch. Therefore, now I have much more personal insights on the importance of being capable of doing independent research. This is, from my point of view, an indispensable trait to become a competent researcher. Even in the context of industry, in order to have effective and efficient communication with colleagues, one must do his or her own homework to be on part with others' understanding. Otherwise, the communication would be just one way and it is never going to be effective.

References

[1] Pepper, A., Tischler, N., and Pryde, G.J. (2019). "Experimental realization of a quantum Autoencoder: The compression of qutrits via machine learning. " Phys. Rev. Lett., 122, 060501.

[2] Bondarenko, D., and Feldmann, P. (2020). "Quantum autoencoders to denoise quantum data." Physical Review Letters 124 (13): 130502.

[3] Nielsen, M.A., and Chuang, I.L. (2010). Quantum Computation and Quantum Information. (10th anniversary ed.). Cambridge: Cambridge University Press. ISBN 978-1107002173. OCLC 665137861.

[4] Beer, K., Bondarenko, D., Farrelly, T., Osborne, T., Salzmann, R., and Wolf, R. (2019). "Efficient learning for deep quantum neural networks." arXiv preprint arXiv:1902.10445.

[5] Bertsekas, D.P. (1999). Nonlinear Programming: 2nd Edition, Athena Scientific, Belmont, Massachusetts, pp. 187.

[6] Cauchy, A. (1847). "Méthode générale pour la résolution des systemes d'équations simultanées." Comp. Rend. Sci. Paris, 25: 536-538.

[7] Kramer, M.A. (1991). "Nonlinear principal component analysis using autoassociative neural networks." AIChE Journal. 37 (2): 233-243. doi:10.1002/aic.690370209.

[8] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press, Massachusetts. ISBN 978-0262035613.

[9] Vincent, P., and Larochelle, H. (2010). "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." Journal of Machine Learning Research. 11: 3371–3408.

[10] Kingma, D.P., and Welling, M. (2019). "An introduction to variational autoencoders." Foundations and Trends® in Machine Learning, 12(4), pp.307-392.

[11] Liou, C.Y., Huang, J.C., and Yang, W.C. (2008). "Modeling word perception using the Elman network." Neurocomputing. 71 (16–18): 3150. doi:10.1016/j.neucom.2008.04.030.

[12] Hinton, G.E., Krizhevsky, A., and Wang, S.D. (2011). "Transforming auto-encoders." In International conference on artificial neural networks (pp. 44-51). Springer, Berlin, Heidelberg.

[13] Kok, P., Munro, W. J., Nemoto, K., Ralph, T. C., Dowling, J.P. and Milburn, G. J. (2007). "Linear optical quantum computing with photonic qubits." Reviews of Modern Physics, 79(1), p.135.

[14] QETLAB (2016). QETLAB: A MATLAB Toolbox for Quantum Entanglement. Retrieved from http://www.qetlab.com/Main_Page 2 Oct 2019.

[15] Wolf, R., (2019). DeepQNN. Retrieved from https://github.com/R8monaW/DeepQNN 10 Oct 2019.

[16] Wootters, W. K. and Zurek, W. H. (1982). "A single quantum cannot be cloned." *Nature*, vol. 299 (5886), pp. 802-803.

[17] Harvey-Collard, P., D'Anjou, B., Rudolph, M., Jacobson, N.T., Dominguez, J., Eyck, G.A.T., Wendt, J.R., Pluym, T., Lilly, M.P., Coish, W.A., Pioro-Ladrière, M., and Carroll, M.S. (2018). "High-Fidelity Single-Shot Readout for a Spin Qubit via an Enhanced Latching Mechanism." Physical Review X, vol. 8 (2), p. 021046.

[18] Bergstra, J., Breuleux O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). "Theano: A CPU and GPU Math Expression Compiler." Proceedings of the Python for Scientific Computing Conference (SciPy).

[19] Reck, M., Zeilinger, A., Bernstein, H. J., and Bertani, P. (1994). "Experimental realization of any discrete unitary operator." *Physical Review Letters,* vol. 73 (1), pp. 58-61.

[20] Davenport, T.H., and Ronanki, R., (2018). Artificial Intelligence for the Real World. Retrieved from https://hbr.org/2018/01/artificial-intelligence-for-the-real-world 11 April 2020.

[21] Darrah, K. (2019). Investment in UK Quantum Computing Is about to Leap. Retrieved from https://sifted.eu/articles/accenture-quantum-computing 20 April 2020.

Appendix

Data used to generate covariance matrix for PCA

[0.44162798, 0.37197811, 0.81645393],
[0.76329805, 0.64291708, -0.06351158],
[-0.55263355, -0.46547682, -0.691323],
[0.41078333, 0.34599803, 0.84352974],
[0.76431204, 0.64377115, -0.03722651],
[0.75696287, 0.63758103, -0.14316996],
[-0.10240303, -0.08625288, -0.9909965],
[-0.69456555, -0.58502449, -0.41871833],
[0.71087393, 0.59876085, -0.36897655],
[-0.730708 , -0.61546686, 0.2954088],
[-0.64669618, -0.54470468, -0.53392963],
[-0.70896391, -0.59715206, -0.37520606],
[0.68471397, 0.57672662, 0.44559307],
[-0.75906037, -0.63934773, 0.12272671],
[0.29090989, 0.24503002, 0.92484146],
[-0.56061198, -0.47219696, -0.68025307],
[-0.70196616, -0.59125794, 0.39706117],
[-0.64219788, -0.54091581, 0.54313164],
[-0.76219194, -0.64198541, -0.08317562],
[0.45625089, 0.38429482, 0.80259119],
[-0.33334526, -0.28077284, -0.90002641],
[0.35984411, 0.30309251, 0.88240985],
[0.0084806 , 0.00714311, -0.99993853],
[0.69712038, 0.5871764, 0.41139646],
[0.7634453 , 0.6430411, 0.0604104],
[0.66708613, 0.5618789, 0.48917093],
[0.7641721 , 0.64365328, -0.04185029],
[0.70737193, 0.59581116, -0.3803078],
[-0.26336659, -0.22183062, -0.93884462],
[0.37228697, 0.31357299, -0.87354129],
[0.4043636 , 0.34059076, 0.84881565],
[0.74357483, 0.62630444, 0.23417778],
[-0.30774548, -0.25921044, 0.91547947],
[-0.66658681, -0.56145833, -0.49033312],
[0.56770839, 0.47817418, 0.67011688],
[0.52837245, 0.44504198, -0.72302157],

```
[-0.72820486,  -0.61335849,  0.30579248],
[ 0.75093979,  0.63250786,  -0.18979791],
[ 0.39544897,  0.33308207,  -0.85596522],
[-0.1270239 ,  -0.10699075,  -0.98611252],
[ 0.23326118,  0.19647318,  0.95235892],
[ 0.4260654 ,  0.35886994,  0.83047013],
[ 0.2094146 ,  0.17638749,  0.96178635],
[ 0.76291963,  0.64259834,  0.07085913],
[ 0.03767865,  0.03173629,  0.99878583],
[ 0.73100569,  0.6157176,  -0.29414709],
[ 0.69215192,  0.58299152,  -0.42549571],
[ 0.06969638,  0.05870445,  -0.99583945],
[ 0.62544627,  0.52680613,  0.57557993],
[ 0.15340826,  0.129214,  -0.97967834],
[ 0.75896128,  0.63926427,  -0.12376982],
[ 0.74540127,  0.62784283,  -0.22403196],
[-0.75359741,  -0.63474634,  0.17084502],
[ 0.24412267,  0.20562169,  0.94769396],
[-0.61325387,  -0.51653661,  0.59758649],
[ 0.66104475,  0.55679031,  0.50299542],
[-0.65573409,  -0.5523172,  -0.51474121],
[ 0.76409682,  0.64358987,  0.0441377 ],
[ 0.75834516,  0.63874531,  -0.13006554],
[ 0.19782561,  0.16662621,  0.96597139],
[ 0.34478705,  0.29041013,  0.8926275 ],
[ 0.35650431,  0.30027944,  0.88472422],
[-0.37100907,  -0.31249663,  0.8744702 ],
[ 0.75807533,  0.63851804,  0.13272715],
[ 0.28266285,  0.23808364,  -0.92920283],
[ 0.69342485,  0.58406369,  0.4219379 ],
[ 0.76259163,  0.64232207,  0.0766574 ],
[-0.04411986,  -0.03716165,  -0.99833484],
[ 0.50666214,  0.42675563,  0.74911489],
[-0.34438774,  -0.2900738,  -0.89289096],
[ 0.75318149,  0.63439602,  0.17395211],
[ 0.58751735,  0.49485904,  0.64026393],
[ 0.37794718,  0.31834052,  0.86937635],
[-0.37853272,  -0.31883371,  -0.86894076],
[-0.67468944,  -0.56828307,  -0.47100797],
[-0.71528373,  -0.60247518,  -0.35410851],
[ 0.51479877,  0.43360902,  -0.73957112],
[-0.32172384,  -0.27098425,  -0.90722726],
```

```
[ 0.71131167,   0.59912955,   0.36753161],
[ 0.31882997,   0.26854678,   -0.90897199],
[-0.22352825,   -0.18827524,  0.95634071],
[-0.73757012,   -0.62124674,  0.26465599],
[-0.26050728,   -0.21942226,  0.94020733],
[ 0.53688958,   0.45221585,   -0.71221514],
[-0.68288847,   -0.57518902,  0.45035645],
[ 0.72800797,   0.61319265,   0.30659284],
[ 0.62735634,   0.52841496,   0.57201543],
[-0.30341465,   -0.25556263,  -0.91794733],
[-0.0617303 ,   -0.05199472,  -0.99673764],
[ 0.75535214,   0.63622433,   -0.15704059],
[ 0.16093562,   0.1355542,    -0.97761178],
[-0.11982386,   -0.10092625,  0.98765183],
[ 0.61224929,   0.51569046,   -0.59934478],
[-0.41173086,   -0.34679612,  -0.84273967],
[-0.2366746 ,   -0.19934826,  0.95091819],
[-0.73308912,   -0.61747245,  0.28514577],
[ 0.74942345,   0.63123067,   0.19978072],
[ 0.19593572,   0.16503438,   -0.96662963],
[-0.11141634,   -0.09384468,  -0.98933289],
[-0.14284291,   -0.12031492,  -0.98240533]
```