CHARACTERIZATION AND EXPERIMENTAL IMPLEMENTATION OF A QUANTUM PERCEPTRON ON A QUANTUM PROCESSOR

A Thesis

by

ANDREW SCHALL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	H. Rusty Harris
Committee Members,	Jun Zou
	Alexey Akimov
	Peng Li
Head of Department,	Miroslav M. Begovic

December 2019

Major Subject: Electrical Engineering

ABSTRACT

Quantum computing and machine learning are both promising technologies that have seen rapid progress recently. However, the state of the art in these fields suggests it would be advantageous to combine both technologies to efficiently and elegantly extract the greatest degree of utility out of both. This work investigates a specific algorithm for a quantum perceptron, initially proposed by Yamamoto.

This work covers the simulation and characterization of the quantum perceptron, novel implementation of quantum bias and physical implementation of the algorithm. Most simulations were run using the QuTiP python library and served to provide feedback and insight into the operation of the perceptron. This was especially important in lieu of a sophisticated training algorithm or rigorous mathematical models of the algorithm. Simulations were run to obtain relationships between inputs and outputs as weight values were changed. The bias was implemented via an auxiliary perceptron whose input value was independent of the input data values. This solves the null-input, null-output issue perceptron issue. However we found that implementing the bias in this manner greatly increases the dimensions of probability space in which the perceptron can perform calculations. Furthermore, this method gives access to data of how the algorithm would expand, specifically how one quantum perceptron would interact with another.

As general-purpose quantum computers become more available to the public, testing theoretical architectures becomes possible. The maturing of the experimental aspect of this field allowed for the quantum perceptron to be tested on IBM's Quantum Experience cloud platform. Not only did this provide a second simulator with which to verify our theoretical results, more

importantly, this gave access to a physical quantum computer to run the algorithm. Current generation quantum processors are noisy, losing information fidelity to the environment due to decoherence. This presented the challenge of designing lean quantum circuits to represent the algorithm in order that the information input into the processor would have the greatest chance of processing and accurately measured before it decays. Using this service, it is shown that the quantum perceptron is functional, practical and easily expandable on real quantum hardware.

DEDICATION

To my family…

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr. Harris, for this exciting research opportunity. I am grateful for the time he has invested in me and guidance that has proven invaluable over the past two years.

I would also like to thank my committee, Dr. Li, Dr. Zou and Dr. Akimov for taking time out of their schedules to review my materials and attend my defense.

Thank you to Alexandre Yamamoto who laid the foundation for this project and made himself readily available anytime I had questions. This project would not have been successful without his assistance. Furthermore, I would like to thank the rest of the lab members, Michael, William, Jung Hwan, Chen, Abhishek and Kastubh for creating a supportive environment to cultivate ideas.

Finally, I would like to thank my wife for her constant uplifting presence and my family for supporting me in all of my life's endeavors.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of my advisor Professor Harris of the Department of Electrical Engineering, Professor Li of the Department of Electrical Engineering, Professor Zou of the Department of Electrical Engineering and Professor Akimov of the Department of Physics.

Alexandre Yamamoto of the Department of Electrical Engineering provided the initial QuTiP code for quantum perceptron simulation at the beginning of the project, as well as gave valuable insight and advice that allowed for its smooth development over time.

All other work conducted for the thesis was completed by the student, under the advisement of Professor Harris of the Department of Electrical Engineering.

Funding Sources

No outside funding was received for this research or writing of this document.

NOMENCLATURE

QP	Quantum Perceptron
ANN	Artificial Neural Network
AN	Artificial Neuron
QANN	Quantum Artificial Neural Network
QC	Quantum Computing
ML	Machine Learning
IBM QX	IBM Quantum Experience
QI	Quantum Information
SIS	Superconductor-Insulator-Superconductor
QFT	Quantum Fourier Transform
IQFT	Inverse Quantum Fourier Transform
NP	Nondeterministic Polynomial

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGEMENTS v

NOMENCLATURE vii

TABLE OF CONTENTS viii

LIST OF FIGURES x

LIST OF TABLES xi

CHAPTER I INTRODUCTION 1

1.1 Introduction to Machine Learning 2
1.2 Introduction to Quantum Computing 5
1.3 Introduction to Quantum Machine Learning 7
1.4 Outline of Thesis 8

CHAPTER II QUANTUM PERCEPTRON 9

2.1 Perceptron Overview 9
2.2 Quantum Perceptron Architecture 10

CHAPTER III SIMULATION AND EVALUATION METHODOLOGY 17

3.1 Quantum Perceptron Simulation 17
3.2 Training 17
3.3 Quantum Information Mapping 19

3.4 Wigner Analysis 23

CHAPTER IV IBM QUANTUM EXPERIENCE EXPERIMENTS AND HARDWARE IMPLEMENTATION 27

4.1 IBM Quantum Experience 27
4.2 Quantum Perceptron Experimental Application 30
4.3 Training Pattern Experimental Results 32
4.4 Quantum Image Processing 38

CHAPTER V CONCLUSION 44

REFERENCES 46

LIST OF FIGURES

FIGURE		Page
1	Diagram of Generic ANN	3
2	Diagram of Bloch Sphere	6
3	Classical Perceptron Diagram	9
4	Node Diagram of QP	11
5	Quantum Circuit Diagram of QP Size N	12
6	Custom Bias Gate Representation	16
7	QI Map for 2-Neuron QP with 2 Input Qubits per Neuron	20
8	Expanded QP QI Map for 3-Neuron, 3 Qubit per Neuron Case	22
9	Wigner Functions of Input States of QP	24
10	Wigner Functions of Bias States of QP	24
11	QP State Relationships Shown Using Wigner Functions	26
12	IBM Quantum Processor Overview	30
13	Experimental Circuit Configuration for QP	31
14	Experimental vs Simulation Results for Input-Dependent Training Pattern	32
15	Experimental vs Simulation Results for Bias-Dependent Training Pattern	34
16	Experimental vs Simulation Results for XOR Training Pattern	35
17	Experimental vs Simulation Results for Diagonal Gradient Training Pattern	36
18	QI Map Tracking the Weights of the Experimental Results	38
19	Basic Scheme for Quantum Image processing Using a QP	40
20	Experimental Example of Quantum Image Processing with QP	42

LIST OF TABLES

TABLE Page

1 Speedup of Quantum Machine Learning Subroutines 8

2 Horizontal Pattern Weight Summary 33

3 Vertical Pattern Weight Summary 34

4 XOR Pattern Weight Summary 35

5 Diagonal Gradient Pattern Weight Summary 37

CHAPTER I

INTRODUCTION

As the end of Moore's Law becomes increasingly imminent, alternative technologies for computing solutions will be paramount to pushing industry forward. Two areas of interest to fill this need are Machine Learning (ML) and Quantum Computing (QC). This paper will investigate the implementation of a quantum perceptron, as well as methods to characterize its function.

At the crux of increases in computational power has been the ability of microchip manufacturers to continuously minimize the area on a silicon wafer needed to construct a transistor. Industry has enjoyed decades of rapid development in this area, but recently progress has waned as transistor sizes are on the order of atoms [6, 26, 35]. While clever architecture techniques have helped alleviate this barrier somewhat, this does not change the fact that physical limits are being reached beyond which conventional transistor fabrication cannot be pushed any further. Another problem that goes in hand with this is a computer science related problem of algorithm efficiency. Classical algorithms have struggled to reach or exceed polynomial time efficiencies when dealing with NP class problems [5]. Especially with the rise of big data, this has created a compounding issue where the accepted solution has simply been to dedicate exponentially more hardware to a problem.

Quantum calculations break this boundary in calculation speed and pave way for new, elegant algorithmic solutions. The most famous example is the result by Shor showing a polynomial time algorithm for prime number factoring, which arguably ignited the QC phenomena ongoing today [29, 30]. This had wide implications, such as potentially rendering RSA security encryption obsolete as it relies on the fact that prime number factoring currently takes exponential

time on classical hardware. Other notable cases are the Grover search and the Deutsch algorithm. The Deutsch-Jozsa problem is more proof of concept, being a simple multi-bit input, single bit output problem with a black box oracle mediating the calculation. It is deliberately designed to favor quantum calculations over classical in order to create a problem whose solution can be reached accurately in polynomial time using QC methods but requires exponential time to solve classically [9]. While not being particularly practical in and of itself, it laid the groundwork leading to Grover and Shor's work later on. Grover's work shows that using QC could enhance database searches from the classical limit of $O(N)$ to $O(\sqrt{N})$ where N is the size of the database [12, 13].

This research builds on the foundation for a modular Quantum Perceptron (QP) from previous work in [27, 28, 37] that scales easily to large problems. The improvements presented here include being able to enact the algorithm on existing quantum hardware and creating a scheme for utilizing a bias signal to maximize utility of the QP. Aside from implementation, I also put forward methods for interpreting and developing intuition of the operation of the QP, which has not been a very well understood entity. Doing so unveils the rich Hilbert space in which the QP can perform calculations. Ultimately, it would seem that the prospects for advancements in computing are more promising by bringing QC and ML together as QC grants a much more expansive foundation to further build ML algorithms.

1.1 Introduction to Machine Learning

ML is a technique that uses various methods to analyze training data and makes extrapolations based upon that data in order to make calculations on new data. Essentially, it seeks to generate code based on output rather than require code in order to give an output. This work is concerned with a specific, biologically inspired, technology called a perceptron which takes in

data, compares it to output and adjusts a weight function in order to discern a relationship between input and output data [24].

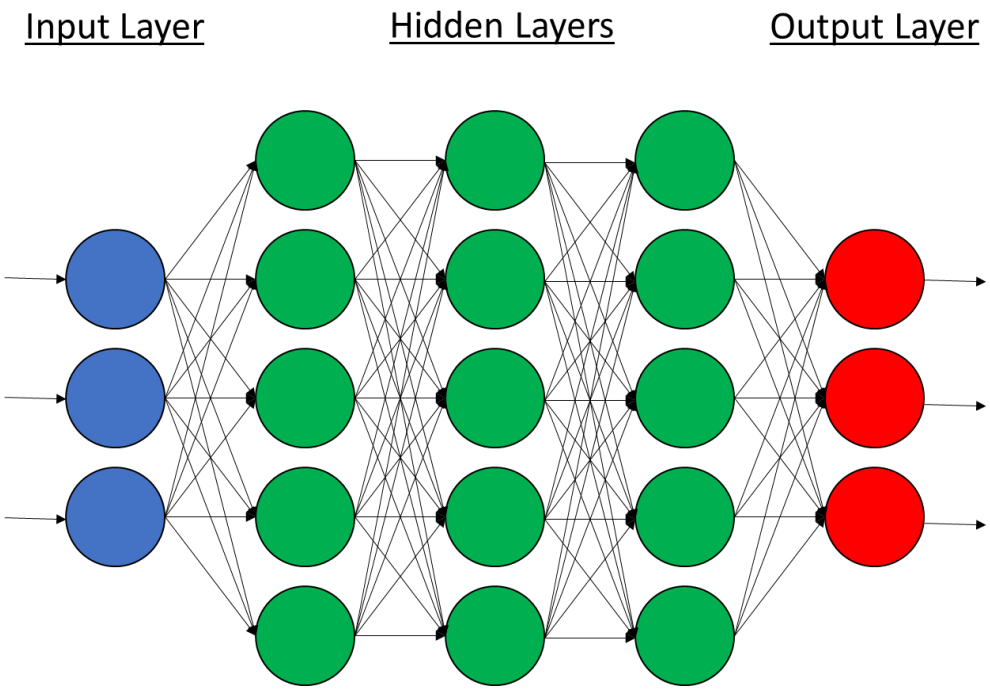

Figure 1: Diagram of a generic ANN. Each circle represents a node and the arrows, also called edges, represent relationships between nodes. Weights are associated with these edges and are actively adjusted in order to achieve a desired input-output relationship. The blue circles are input nodes, which feed input signals into the network. Green circles are hidden nodes which add extra degrees of freedom to a given calculation. Generally speaking, the more hidden layers a network can take advantage of, the more complex classifications it can perform. Finally, the red output layer allows the user to read the result of the network's calculations.

The human brain has long been a subject of mystery and admiration for its remarkable ability to process data. Not only does it take in known information and store it, but it has the ability to generate new information that did not exist before based on what it is given and does so incredibly efficiently [33]. This has spawned the field of thought for design known as Artificial Neural Network (ANN), a sub-model within machine learning, which seeks to design computing architectures taking inspiration from biological neural systems. The hallmark of the ANN by which it seeks to obtain this goal is by creating a massively, more or less randomly, interconnected network of relatively simple nodes referred to as neurons, however I will refer to them as artificial neurons (AN) in this section for clarity in order to distinguish them from their natural counterparts. The AN, which will be the main focus of this work and be discussed in greater detail later, as the fundamental component of the network must be able to accept, store and transfer information. This generally follows closely to the natural model of how neurons are structured to receive and send electrochemical signals. Dendrites take signals into the cell, through the soma containing the nucleus which leads to the axon which terminates in synapses. The synapse, upon receiving a signal, will then fire or lay dormant according to an action potential, a threshold adjusted by experience, which determines whether or not information will be passed to the next neuron. Similarly, AN's have input nodes to take in signals and weight nodes to determine the relevance of each AN to a specific calculation. The dendrites and axon terminals facilitate many input and output connections between neurons which inspires the large interconnected nature of ANN's.

Arguably, one of the most astounding properties of the brain is its neuroplasticity and it is precisely this aspect that we wish to replicate in the ANN's if we want them to be useful for more than one task and possess an advantage over traditional programming. This capability is accounted for at the AN level and is accomplished by adjusting the weights. Similar to in a neuron, these

weights are adjusted in accordance to feedback from events that manipulate the frequency with which the AN is active. This work will focus primarily on supervised training, which enacts these changes in a training loop that presents the network with a small subset of data containing labels for problems and solutions; ideal input and output signals. The feedback from this workflow is what is used to adjust the weights of the network and ultimately has the goal of shaping the network to best fit the particular data set. After allowing the network to "practice" a certain number of these training subsets, the network will then be given data outside the subset, with the goal being for it to arrive at the correct solution despite it not having trained on these specific data points previously. The ability of a network to classify novel data is called generalization and exhibiting this quality makes an algorithm valuable.

1.2 Introduction to Quantum Computing

QC was first theorized in the 1980's and seeks to take advantage of physical quantum phenomena in order to perform more complex calculations [10, 11, 15]. The advantages of QC that will be primarily discussed here are information density, entanglement and phase. This technology is manifest in two state quantum systems that can be relatively easily manipulated, such as trapped ions, quantum dots and superconducting oscillators. The generic name for these systems is the qubit, the unit of calculation used in QC, which is represented as a vector anchored at the origin of a unit sphere, called a Bloch Sphere. This allows one to treat the system more like a spherical state machine, navigating the sphere based on permitted geometrical transformations in order to accomplish useful calculations. The poles of this unit sphere are pure states, representing classical 1 and 0, however if the vector points elsewhere, it is a weighted combination of the two states.

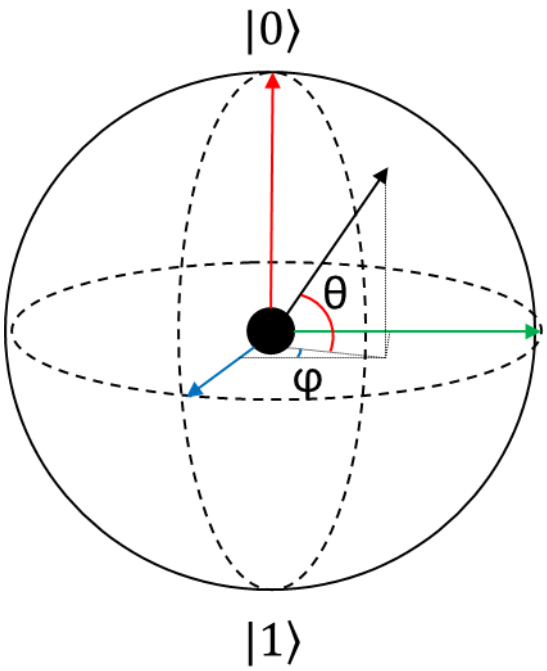

Figure 2: Diagram of Bloch sphere. The state of the qubit can be described by the two angles (θ,φ). The state vector is restricted to traveling along the surface of the sphere and the closer it is to one pole, the more likely it is to produce that pole upon measurement. E.g. the state vector in the northern hemisphere is more likely to produce a |0⟩ upon measurement. It is worth noting that, while this seems to imply θ is more impactful than φ, many quantum operations function based on axes exchange, so these values can be exchanged throughout an algorithm.

This weighted combination translates to a weighted probability which resolves to a pure state upon measuring according to the probabilities previously mentioned. This ability of probabilistically representing two states with a single computational object leads to the information density advantage. Entanglement and phase are both uniquely quantum phenomena, phase being an intrinsic property of quantum objects and entanglement being a description of interacting quantum

objects. Phase is the space a quantum object can occupy between pure states, allowing for an extra degree of freedom in calculations. Entanglement, on the other hand, is the phenomena by which the measurable of two or more quantum objects are correlated. This allows for a global calculation to take place in some sense as, depending on the architecture, there can potentially be an all-to-all dependence between qubits, and thus instantaneous global calculations can be performed.

1.3 Introduction to Quantum Machine Learning

There has been exciting research recently showing that if these two technologies are properly leveraged, quantum computers, and ideally by extension the QP, is a prime candidate for hosting machine learning technologies moving forward. The basic argument stems from the fact that quantum computers can much more easily produce computationally complex patterns in comparison to classical machines. If quantum systems can produce these rich patterns, then with the right algorithms, perhaps they can be trained with greater ease than their classical counterparts [1, 4]. Although it is an open question as to what constitutes an objective improvement when it comes to quantum augmentation, particularly due to the fact that it is so difficult to define calculation efficiency in ML to begin with, the preliminary evidence is promising that we are moving in the right direction towards useful quantum algorithms. For the development of near- and long-term applications, it is also convenient that quantum calculations can be performed on not only Quantum Information (QI), but classical information as well. This means that purely quantum solutions do not need to be developed in order to take advantage of quantum computers and that hybrid quantum solutions can be appended to existing classical solutions for advantages that can be seen today [4]. However, these higher-level architectures will need more time to

develop but the aim of this paper is to understand a more fundamental technology that will lead to the realization of these sorts of advancements in the future.

Table 1: Speedup of Quantum Machine Learning Subroutines

Subroutines	Speedup
Bayesian Interference	$O(\sqrt{N})$
Online Perceptron	$O(\sqrt{N})$
Quantum Boltzmann Machine	$O(\log(N))$
Quantum Support Vector Machine	$O(\log(N))$
Divisive Clustering	$O(Nlog(N))$

[4]

1.4 Outline of Thesis

The primary goals of this work are to bring to fruition a theoretical algorithm for a QP by porting it to physical quantum hardware as well as determining a technique for developing intuition of the algorithm. Chapter two covers an overview of this algorithm, how it relates to classical perceptron and how the quantum circuit functions mathematically and given infinite resources. The data covered in chapter three will be based on the QuTiP simulations and cover QP function as well as quantum output maps which will help characterize the system. Chapter four on the other hand, will cover the IBM experimental data, demonstrating the architectural changes necessary to turn theory into experiment and the effects decoherence have on quantum systems. Finally, this paper will be concluded in chapter 5 which will consist of final remarks and some insight into future work.

CHAPTER II

QUANTUM PERCEPTRON

2.1 Perceptron Overview

The perceptron is an algorithm first formalized by Rosenblatt in 1957 as a means for processing data as inspired by the biological neuron [24].

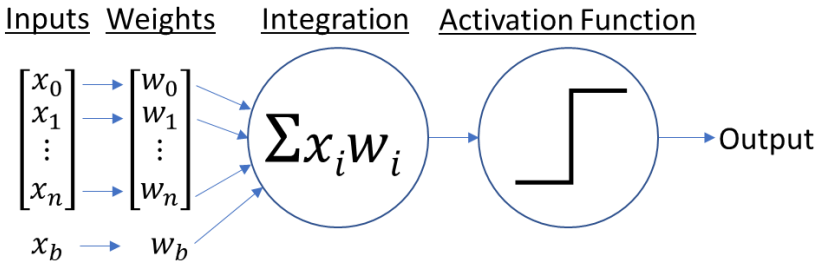

Figure 3: Classical perceptron diagram. The input vector and weight vector are integrated and compared to an activation function. The perceptron is an all-or-nothing algorthm, so the integration will either be greater than the threshold and give an active output or otherwise it will remain dormant.

As we know it today, perceptrons are generally constructed from four fundamental components; inputs, weights, the integration function and a nonlinear activation function. The goal of the structure is to automate mapping of outputs to inputs, which is based on the adjustment of weights. Weights are adjusted in a training loop, designed to teach the perceptron to discern patterns within the data. Typically, perceptrons are trained in a supervised training loop, with the ultimate goal being to complete training without loss of generality. However, the perceptron is

limited in that it is a linear classifier, capable only of separating data with a hyperplane. This means that classifying nonlinear data, or even just data with a low degree of separation, can be impossible with perceptrons. However, it has been shown that creating an ANN from perceptrons permits them to classify any arbitrary data set, where the only limiting factor is the size of the network [7, 20].

The input of the perceptron defines the dimensionality of the problem and the space where data can reside. The inputs and weights are convolved into a summation function which is compared to a nonlinear output function, usually a step function or sigmoid function of some kind, in order to ascertain the desired outcome of the algorithm. This output function behaves as a threshold, an input-weight combination producing a signal that exceeds this minimum will cause the perceptron to become active and a lower signal will cause it to become inactive.

$$sigmoid(0) > \sum_{0}^{i} x_i w_i \rightarrow f(x,w) = 0 \quad (1)$$

$$sigmoid(0) < \sum_{0}^{i} x_i w_i \rightarrow f(x,w) = 1 \quad (2)$$

2.2 Quantum Perceptron Architecture

An alternative solution resolving the perceptron's limited classification abilities is constructing a quantum implementation of the classical algorithm [20, 27, 28]. This work will focus on building on previous architecture put forth in [37], taking the theoretical groundwork and applying it in an experimental setting. In order to accomplish this, the same basic structure

is maintained from the classical version. The necessary components to accomplish this will include: a medium for receiving inputs, a method for storing and updating weights, an integration function to combine the inputs and weights, an activation function to automate information transfer decisions and a method to output information. As the QP relies on classical computers for control of its inputs and outputs, this is somewhat of an exercise in converting between classical and quantum information.

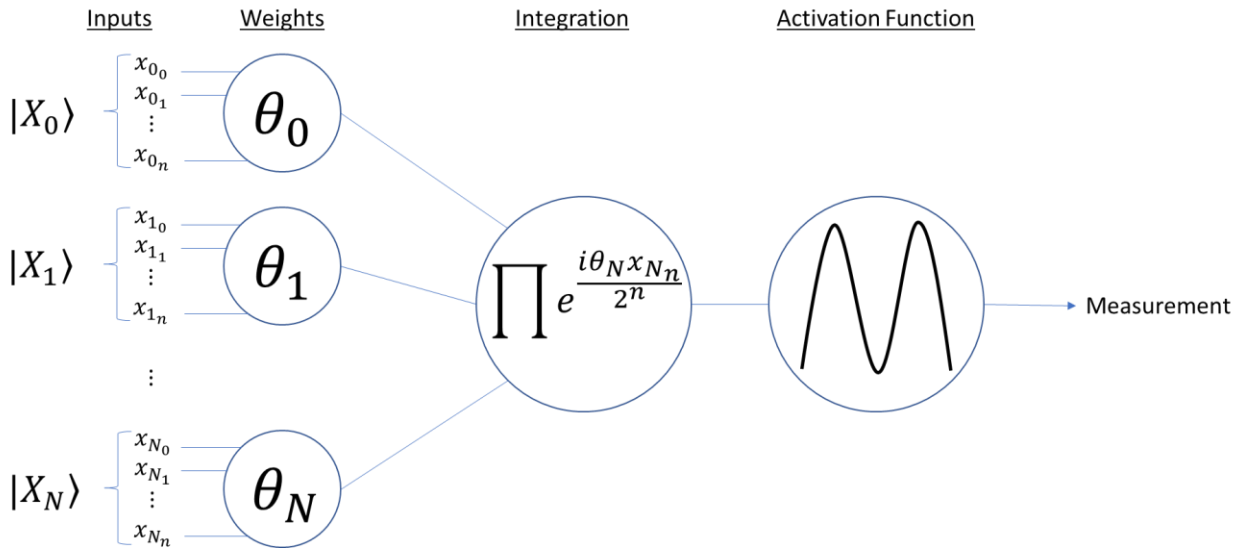

Figure 4: Node diagram of QP. X_N represents the state of each neuron where as x_{Nn} labels the individual qubit states making up the neurons where N signifies to which neuron the bit belongs and n specifies the significance of the bit. The weights, θ_N, carry a similar naming convention, N denoting the neuron the weight is attached to.

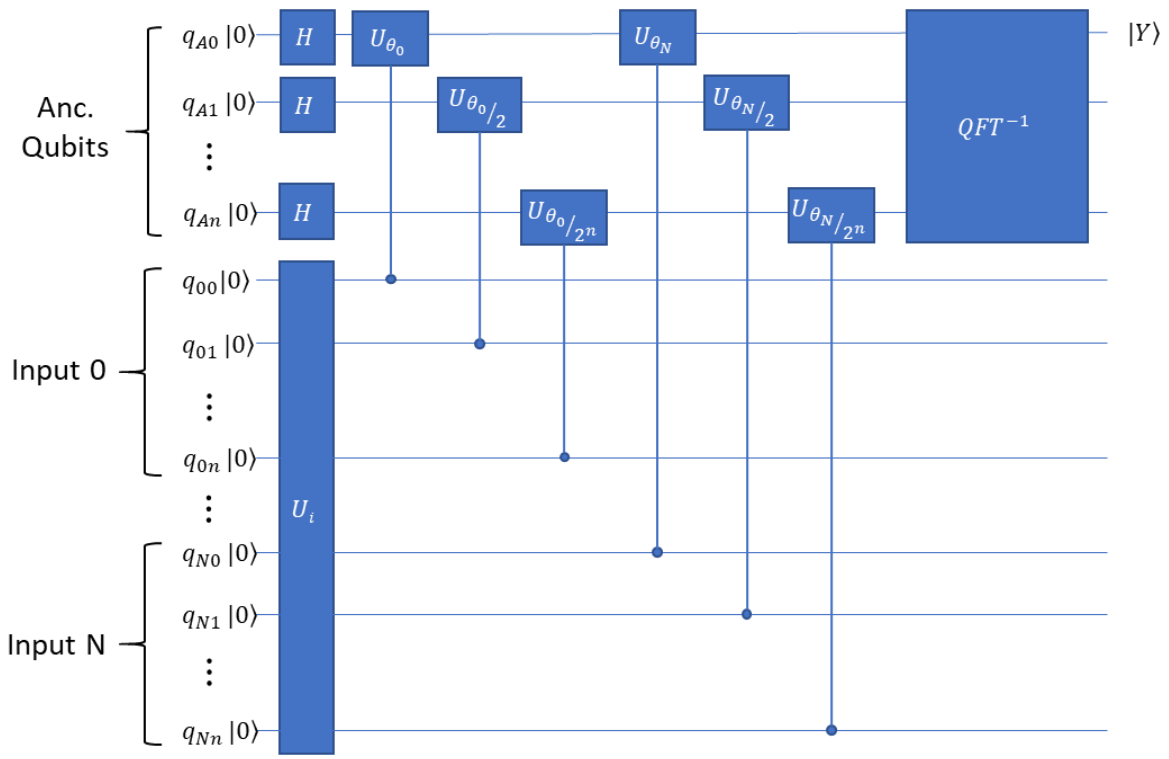

Figure 5: Quantum circuit diagram of QP size N. The U_i gate prepares inputs by placing Pauli-X gates on qubits where the input should be $|1\rangle$, and leaving qubits where input should stay $|0\rangle$. It also becomes clear that most of the calculations are mediated by the ancillary qubits while the input qubits facilitate input signals. The QFT^{-1} gate is the inverse quantum Fourier transform which estimates the overall phase of the system.

The input of the QP functions based on taking a single point of binary data transmitted from a classical computer and encoding it as a pure quantum state. This is done by using the Pauli-X gate which performs a rotation of π radians around the x axis of the Bloch sphere.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3}$$

When acting on pure states, this has the effect of changing a $|0\rangle$ state to a $|1\rangle$ state and vice versa.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |1\rangle \tag{4}$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |0\rangle \tag{5}$$

These X gates can be used to encode our binary data in quantum states on the qubits that we have designated to take the inputs (see figure 5) by selectively placing X gates on qubits we wish to activate to designate it as a "1" and excluding it to designate a "0". In quantum processors, it is standard for all qubits to start out in the ground state and we can represent placement of X gates by a unitary transform, U_i.

$$|\psi_i\rangle = |0\rangle^{\otimes N} \tag{6}$$

$$U_i|\psi_i\rangle = |X_0\rangle \otimes |X_1\rangle \otimes \cdots \otimes |X_N\rangle \tag{7}$$

Utilizing this method gives the QP a way to convert digital binary signals into physical information easily in a way that scales well with the size of the QP, requiring at most only a single gate depth-wise.

Once the input states are created, they can then be used as control signals for the controlled phase gates, which mediate the weight function. A controlled gate operation implies that two or more qubits are entangled in an operation, meaning the states of the entangled qubits affect the outcome of the calculation. In the QP, the controlled phase gates connect input qubits with ancillary qubits. The pure states of the input qubits behave as test signals for the gate, signaling an "on" state for an input of $|1\rangle$ and an "off" state for and input of $|0\rangle$. The ancillary qubits, which are initialized to a superposition state via a Hadamard gate, record changes in phase information by allowing rotations of θ around the equator. When an input is $|0\rangle$, no rotation is recorded, when an input is $|1\rangle$, the rotation is enacted.

$$U_\theta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix} \tag{8}$$

In this scheme, θ is the weight of a given input, but the weight is distributed over the input qubits according to:

$$\theta_N = \frac{\theta}{2^N} \tag{9}$$

Where N is the significance of a given qubit. This is an important feature as there is no inherent difference between individual input signals and it would be wasting input space if the perceptron interpreted different inputs as being identical. For example, if a perceptron had two inputs, if it were only receiving θ as the weight signal, then it could only tell when the system was off, when

an input was on, and when both inputs are on. However, it would not be able to tell the inputs apart from one another due to the fact that they would be transmitting the same signal. This is an example of a scheme that ensures each input qubit will differentiate itself by sending a unique input signals from another input qubit for any given θ.

An important feature of the QP is the ability to use an input neuron as a bias signal. Structurally, the bias input is no different from any of the regular inputs, but its input state is another variable to be controlled by the algorithm rather than part of the data being input. This is an important distinction from [37] as this will allow the QP to be implemented experimentally. Not only does this take care of the null in, null out problem inherent to perceptrons, but it can also participate in the calculation by increasing the Hilbert space in which the QP can search for solutions. Ideally, the maximum number of qubits could be allocated to the bias input, which would be the case if there were N ancillary qubits and M bias qubits, then N = M. In this case, the Hilbert space would increase by a factor of 2^M, which is the greatest amount. However, it is not necessary to allocate this many qubits to the bias and not always practical, as qubits are a valuable resource. The minimum number of qubits needed for a functional bias, and what will most likely be practical for the near future until qubits become more plentiful, is M = 1. The simplest configuration is to leave the single bias qubit entangled with the least significant ancillary qubit, leading to a Hilbert space increase factor of 2. A slightly more complex scheme, and the one used in my experiments, is to dynamically entangle the single bias qubit with any ancillary qubit of the user's choosing and have which qubits are being entangled be another degree of freedom for the QP. This will be represented by a custom gate in figure 6. Using this scheme, we can maximize the Hilbert space for our one bias qubit, obtaining a scaling factor now proportional to the number of ancillary qubits, N+1.

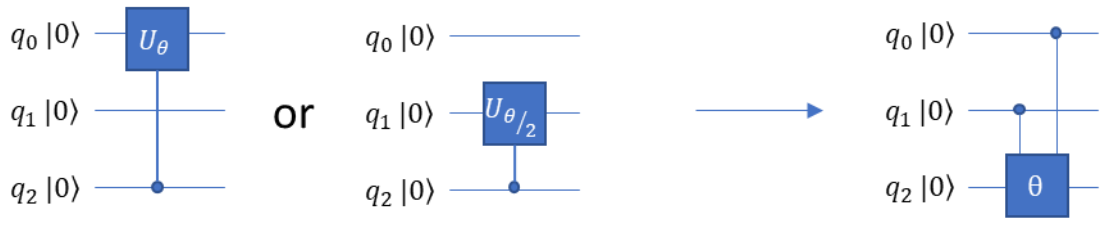

Figure 6: Custom bias gate representation. In order to express the dynamic biasing scheme more simple and concise manner when constructing circuit diagrams, a custom gate will be used. The weight, θ, will be placed on the qubit that handles the bias input signal and the lines connecting it to other qubits show the qubits with which it will have the option of being entangled.

The integration equivalence is built in with the recording of the phase changes in the ancillary qubits as a result of the weight gates, with each phase change compounding on the last.

$$\theta_{Total} = \prod e^{i\theta_N} \tag{10}$$

From the perspective of the Bloch sphere, this results in continuously processing around the equator as more weight is added. This gives the QP a continuous, sinusoidal activation function. The output of the QP is obtained by taking a measurement of the desired ancillary qubit after the Inverse Quantum Fourier Transform (IQFT).

CHAPTER III

SIMULATION AND EVALUATION METHODOLOGY

3.1 Quantum Perceptron Simulation

While the ultimate goal of this project is to implement a QP on quantum hardware, limited access to this technology makes simulation necessary and practical for data collection and prototyping. For this, QuTiP [14] will be utilized which is a python library built to handle quantum mechanics and quantum computing via linear algebra and matrix mathematics. This will allow for processing a higher quantity of data through the QP algorithm much more quickly in order to qualitatively characterize its computation characteristics. One of the ways this will be accomplished is in allowing the QP to be trained, which involves repeatedly creating the quantum system, measuring its state, comparing it to an ideal state, and adjusting weight values as needed until the measured and ideal states are sufficiently similar. This would consume a great deal of resources in an experimental setting and so currently it is much more efficient to train in simulation and simply test the trained weight values on the quantum processor.

3.2 Training

A supervised learning loop is implemented in order to train the QP according to the algorithm in [37]. To accomplish this, neurons will be designated as either input or bias neurons. This defines whether their input values and weights are dependent on the data set, in the case of an input neurons, or if the they process independent values that define extra degrees of freedom for the user to leverage in the bias neurons. In the low dimensional cases to be explored, only

two neuron cases were examined with one being designated an input neuron and the other a bias neuron.

The training set is given by a text file written by the user that defines the input values of both neurons and the desired output. The training progresses according to the following equation:

$$\theta_N^{i+1} = \theta_N^i + \alpha(|T\rangle^i - |Y\rangle^i)x_N^i \tag{11}$$

The weight is updated in the event that the measured output does not match the training output defined by the training set. α is the training intensity value which is a constant that adjusts how much the weight should be modified for an incorrect answer. $|T\rangle^i$ and $|Y\rangle^i$ are ideal training output and measured output respectively. If they match, no change is made to the weight but if they differ then the weight will be updated. x_N^i is the input value for the current training value, which gives higher input values a stronger effect on the training scheme, but also attempts to combat a stagnation in training that giving the same priority to all input values may cause. Especially in the case of limited data sets, one could visualize two training values cancelling frequently, returning +α and - α repeatedly causing the overall system to stall its progress. Finally, θ_N^i is the current weight value and θ_N^{i+1} is the updated weight value after training.

3.3 Quantum Information Mapping

In order to build an intuition for the function of the QP, we mapped the simulated output as the weight was iterated over all values in order to give an overview of all possible values. To elaborate, for this project the QP was restricted to a two-input architecture, one to take in data, $|X_{in}\rangle$, and one to supply the bias signal, $|X_{bias}\rangle$. Each input has its own weight value, θ_{in} and θ_{bias}. By sweeping all configurations means that, for every pair of input values ($|X_{in}\rangle$, $|X_{bias}\rangle$), all combinations of weight pairs ($\theta_{in}, \theta_{bias}$) are iterated from 0 to $2^N\pi$ radians, where N is the number of input qubits, and the output value is recorded and plotted. This periodicity arises from the algorithm used in assigning bit significance. The impact of weight is reduced for less significant qubits which translates to less phase change in the corresponding ancillary qubit. As a result, these neutered qubits will have a longer procession time compared to more significant qubits. The overall periodicity is defined by the least significant qubit as its weight value will be the most impacted by the algorithm, which is why periodicity is proportional to the number of qubits on an input neuron. From these simulations, the primary goal will be to formulate a method for navigating the calculation space of the QP, at least for low dimensional input cases, so that some intuition can be formed about the function of the QP. This understanding of QP operation will then be used to design experimental applications more intelligently.

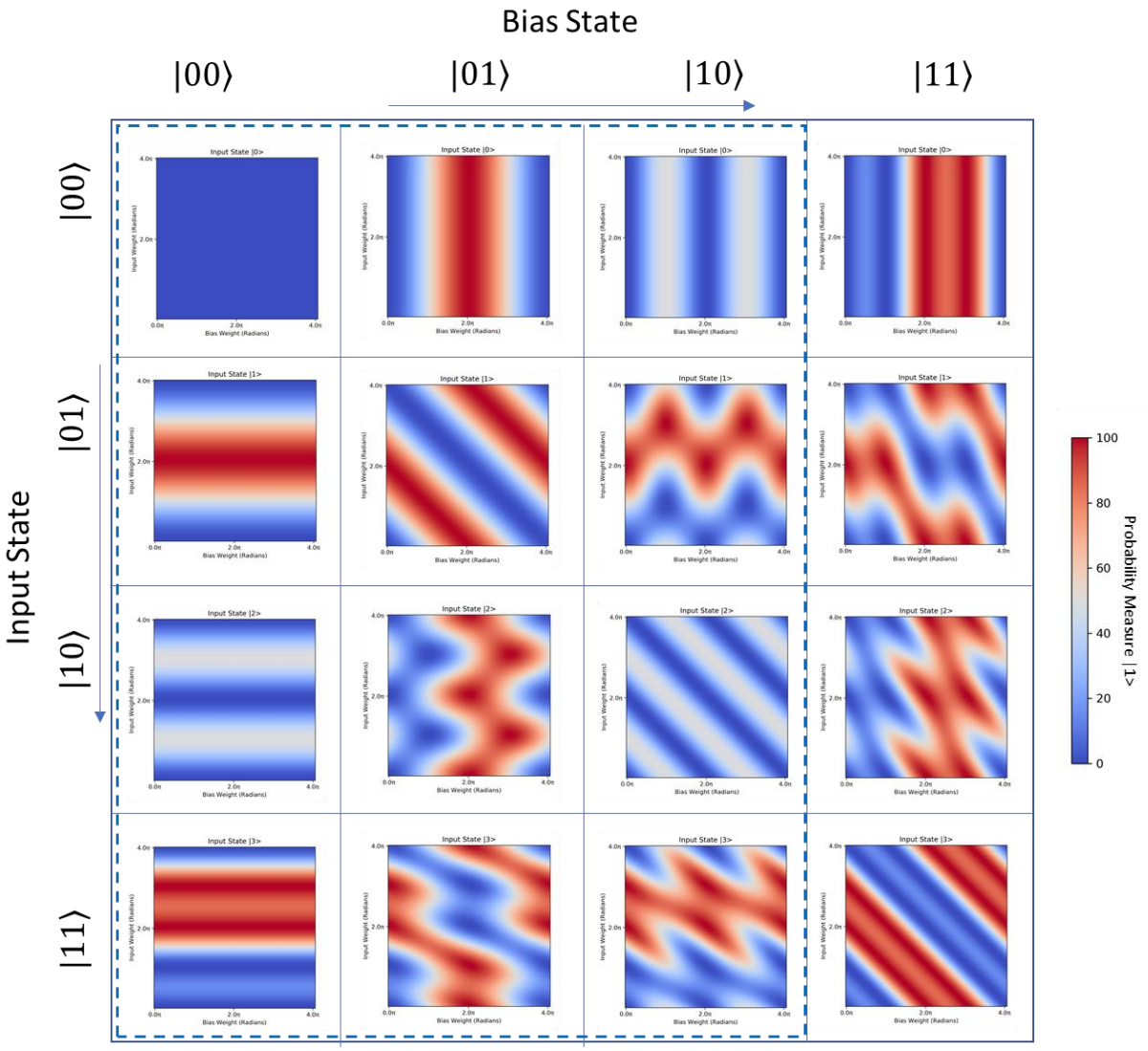

Figure 7: QI map for 2-neuron QP with 2 input qubits per neuron. The maps demonstrate how the atainable solutions possible in output space evolve as the input state combination is changed. The dotted box represents what output space is attainable experimentally as only one qubit could be spared for the bias neuron.

Figure 7 displays an array of QI maps that give an overview of all the solutions the QP can output for any given combination of input state and weight for the case of 2 input neurons

with 2 qubits constructing each neuron. The first row represents the patterns of when only the bias states are active and the input states are inactive, and the first column represents the reverse case. This demonstrates that, for any given input signal, there is a characteristic output pattern that accompanies it. The other patterns map how the output changes when both input and bias signals are active, giving insight into how the base signals convolve with one another. The first notable feature here is the case where the input signals are the same and the resulting output pattern is simply a phase shifted version of the contributing patterns, the column one and row one patterns. The second is the general trans-diagonal symmetry displayed by the system. All of the patterns are mirrored across the diagonal and rotated 90^{o}.

The full, four-by-four output space is the result of simulating all possible input signal combinations for the two input qubits of each neuron of the QP. This requires a total of six qubits, two for the ancillary, two for the input neuron and two for the bias neuron. However, the processors on the IBM QX we elected to use only offer 5 qubits, and so only one qubit could be spared for the bias neuron. This allows it to be turned off, column one, and dynamically entangled between ancillary qubits one and two, columns two and three. This reduced calculation space is represented by the doted box in the figure and will be reflected in the data of Chapter 4.

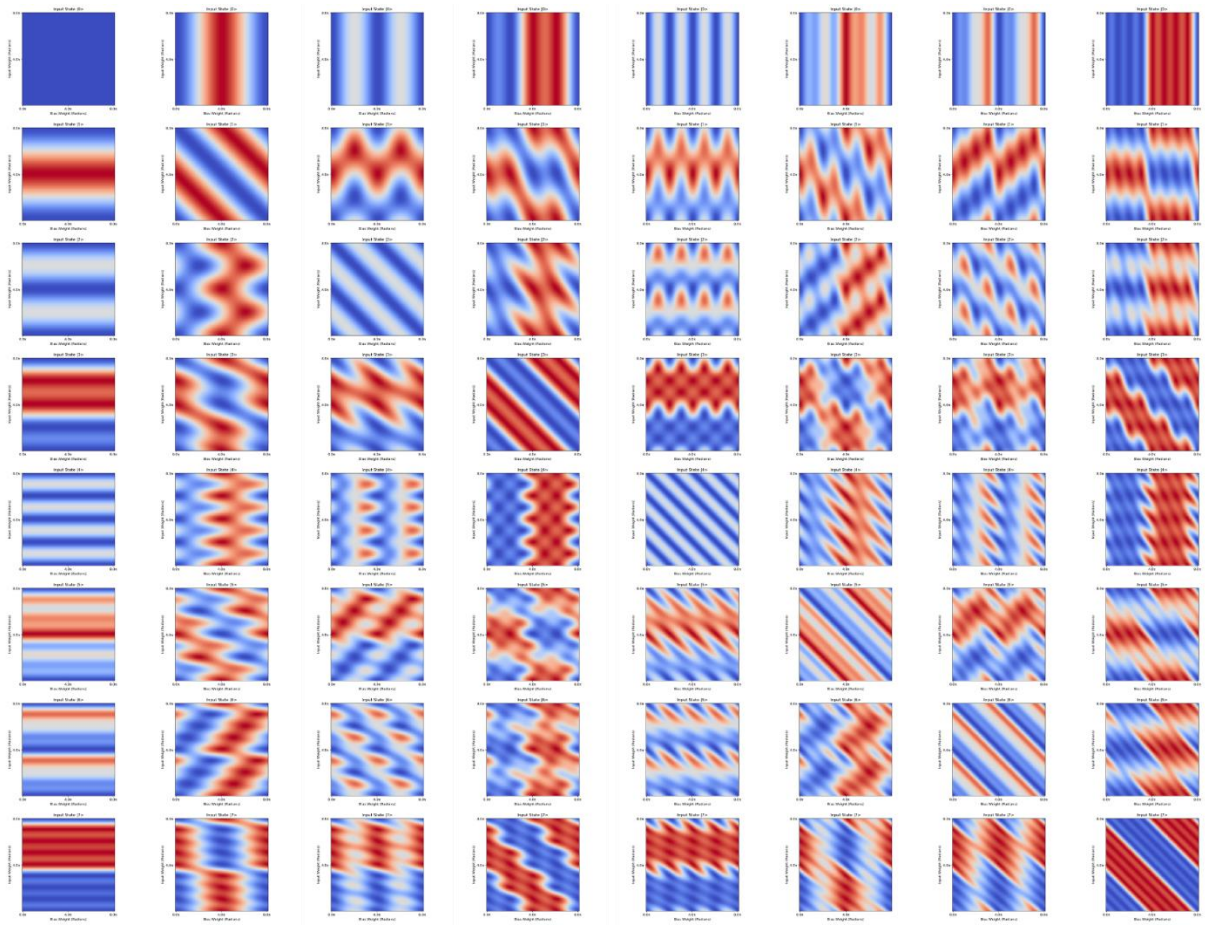

Figure 8: Expanded QP QI map for 2 input neurons, 3 qubits per neuron case.

This figure further extends the QI map to the three-input case. This gives an eight by eight array as well as a periodicity of eight pi in each pattern. These more complex patterns begin to show interesting hallmarks such as localities rather than exclusively continuous regions of $|0\rangle$ and $|1\rangle$ measurements. Additionally, it is demonstrated here that it is possible to combine fundamental patterns that have don't output $|1\rangle$ by themselves, such as $|010\rangle$ and $|100\rangle$, and combine these results to obtain a pattern with regions that output $|1\rangle$. This suggests that states are additive at least to some degree, which implies that an entire dimension cannot be deemed uninteresting because they have "cold", low probability of outputting $|1\rangle$, constituent states.

3.4 Wigner Analysis

One of the shortcomings in the QI maps are the fact that it is best suited to cases of low dimensionality. Quasiprobability functions were investigated as a solution to overcome this problem. A brief study was performed using the Wigner function to process the output state vector of the QP prior to measurement. This was accomplished using the built-in function in QuTiP that allows this analysis to be performed on any state vector or density matrix.

Quasiprobability functions in general are a method of generating a phase space for two observables of a quantum system, position and momentum for example, by treating the system as a quantum harmonic oscillator. This is done in much the same way one would create a probability space for a classical harmonic oscillator. However, in a quantum system this does not combine well with Heisenberg's uncertainty principle and thus the phase space will exhibit odd behavior such as the probability becoming negative in certain regions, which is why it is referred to as a quasiprobability space. The Wigner function is the specific quasiprobability function to be examined here and it is defined by the following equation.

$$W(q,p) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ipx} \left\langle q - \frac{x}{2}\middle|\hat{\rho}\middle| q + \frac{x}{2}\right\rangle \tag{12}$$

The function operates on either a density matrix or state vector, $\hat{\rho}$, in the dimensions of q and p which are quadratures, the in-phase and out-of-phase components of the complex amplitude of the phase space.

$$q = \sqrt{2}(\hat{a}^{\dagger} + \hat{a})\,; p = i\sqrt{2}(\hat{a}^{\dagger} - \hat{a}) \tag{13}$$

$$\hat{a} = \sqrt{2}(q + ip) \tag{14}$$

Here $\hat{a}$ is the annihilation operator [17].

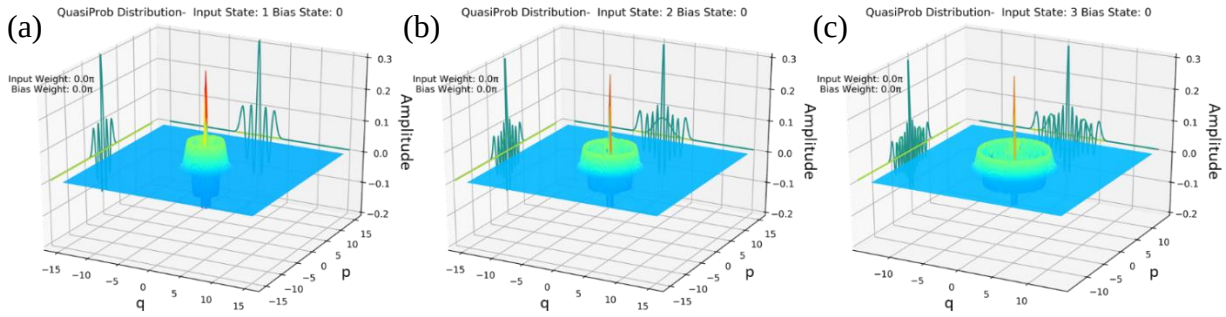

Figure 9: Wigner functions of input states of QP. (a) When the input state is $|01\rangle$ the output gives a Wigner function similar to a Fock state of n = 4. (b) Input state set to $|10\rangle$ generates a Wigner of Fock state of n = 8. (c) Input state $|11\rangle$ gives Wigner of Fock state n = 12.

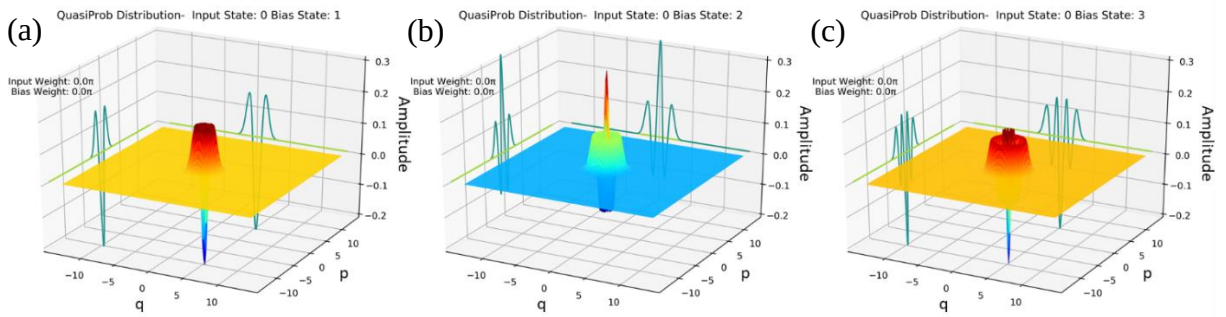

Figure 10: Wigner functions of bias states of QP. (a) When the bias state is set to $|01\rangle$, the output gives the Wigner of Fock state n = 1. (b) Bias state set to $|10\rangle$ produces a Wigner of Fock state n = 2. (c) Bias state $|11\rangle$ gives Wigner of Fock state n = 3.

One of the primary advantages of implementing Wigner analysis is that it gives information about physics of the system, rather than simply the quantum information. Ironically, there is a fair amount of information lost by the QI maps in this regard as they only look at the measured output of one qubit out of many contributing to the overall system. The Wigner function is well suited to handling high dimensional data as it simply operates on a state vector or density matrix and will process them regardless of the number of variables controlling the value of those entities. In addition, the output of the Wigner function can be used to assess aspects of a quantum system. For instance, one can measure the negativite volume to use as an indication quality of entanglement between qubits [25]. Furthermore, it begins to allow building relations between states of the QP. In figure 9 and 10, it is significant that the Fock states for input and bias states $|01\rangle$ and $|10\rangle$ sum to the result in state $|11\rangle$, for example in figure 10, n = 1 and n = 2 sum to n = 3. This means that we can use raising operators to combine states $|01\rangle$ and $|10\rangle$ in order to obtain figure $|11\rangle$ as they are essentially states in a quantum harmonic oscillator, as shown in figure 11. This becomes more intricate as weights are iterated, but this is a first step toward a complete mathematical description of the QP.

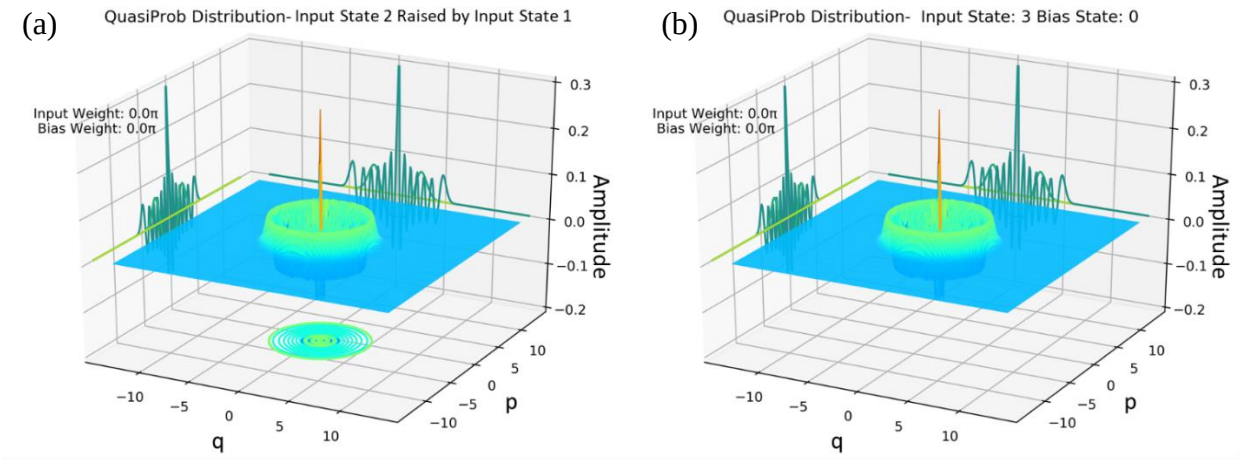

Figure 11: QP state relationships shown using Wigner functions. (a) Wigner function of state vectors of QP from input states $|01\rangle$ and $|10\rangle$ combined using the raising operator. (b) The Wigner function of $|11\rangle$ input state shown for comparison.

However, all of this comes at the cost of no longer being able to generate a look-up style table as a Wigner function can only be generated for specific values of variables, though animations can be used to iterate through a range of values if desired. As both techniques have advantages and drawbacks, using both in combination creates a comprehensive picture of the QP.

CHAPTER IV

IBM QUANTUM EXPERIENCE EXPERIMENTS AND HARDWARE IMPLEMENTATION

4.1 IBM Quantum Experience

In order to test the results of the simulations experimentally, the IBM QX will be used, which is an open access, online platform that allows algorithms to be programmed on quantum hardware and offers several options for data collection. This served as the cornerstone of the experimental setup, providing an ecosystem with a variety of methods for interacting with quantum hardware. For a quick starting point, a visual programming solution is included which allows the user to place gates on lines representing qubits in order to build a quantum circuit. The backend options available to the user in order to run created algorithms are two 5-qubit processor, a 16-qubit processor and a 32-qubit simulator. The simulator is used in order to troubleshoot circuits as users are limited in the number of experiments that can be performed on the quantum hardware. IBM has a credit system that allows space to be "rented" on their hardware, which limit the number of experiments that can be queued at one time as well as the number of experiments that can be run in a day. This is a good tool for low volume experiments and was primarily what was used for training experiments.

Additionally, there is also a framework for controlling IBM's quantum hardware and simulation tools via python using the Qiskit library [2]. It is organized into four primary modules that each facilitate different aspects of the ecosystem. Terra provides the foundation for the quantum computing infrastructure for Qiskit, containing methods for quantum circuit construction, result collection and backend selection. For circuit management, it creates and monitors exchanges of information between quantum and classical registers as well as applies

stored methods for quantum gates and tracks the qubits on which they are executed. It can then create a queue of circuits within a provider, which can target a specific backend, either hardware or simulator in addition to specifying parameters such as number of repetitions. After a job completes, the results can be retrieved by displaying counts for each state that was measured. This can be retrieved as in simple text or with more intricate graphs, such as 3D histograms. Aer contains the local simulation tools, allowing the user to simulate circuits without the cloud as well as some noise simulation tools. The Aqua package grants access to preconstructed algorithms for use in finance or quantum chemistry such as Shor's and Grover's algorithms. Finally, Ignis is a performance-oriented module that is used for analyzing and mitigating error that arises in quantum circuits. Terra and Aer were primarily used for this project, as any algorithms needed were manually coded and no error correction was required.

The IBM processors are constructed of superconducting Josephson junction qubits arranged in a bow-tie configuration. This medium was chosen in part due to its basis in lithographic processes, granting a vast arsenal of mature technologies used widely in the silicon industry to employ in developing this emerging technology. These techniques allow for many devices to be constructed with high precision. Each qubit contains an LC oscillator with nonlinearity introduced by the Josephson junction, a Superconductor-Insulator-Superconductor (SIS) element [31, 36]. As described in [31], the inductor takes part in contributing to the potential energy of the system and the capacitor to the kinetic energy.

$$PE = \frac{\Phi^2}{2L} \tag{15}$$

$$KE = \frac{Q^2}{2C} \rightarrow \frac{p^2}{2m} \tag{16}$$

Where Φ is the magnetic flux, which is proportional to the phase of the system, δ.

$$\delta = \frac{2\pi\Phi}{\Phi_0} \tag{17}$$

$$\Phi_0 = \frac{h}{2e} \tag{18}$$

The nonlinearity of the Josephson junction is due to the modulation of its critical current, I_0, due to phase.

$$E = -I_0 \cos(\delta) \tag{19}$$

This induced anharmonic nature between the ground state and first excited state defines the two states of the qubit, allowing them to be distinguished from other energy levels of the quantum oscillator. The frequency needed to move between states will also be defined by this nonlinearity and Rabi oscillations, typically on the order of GHz, will be used to manipulate the qubits [16, 18, 31].

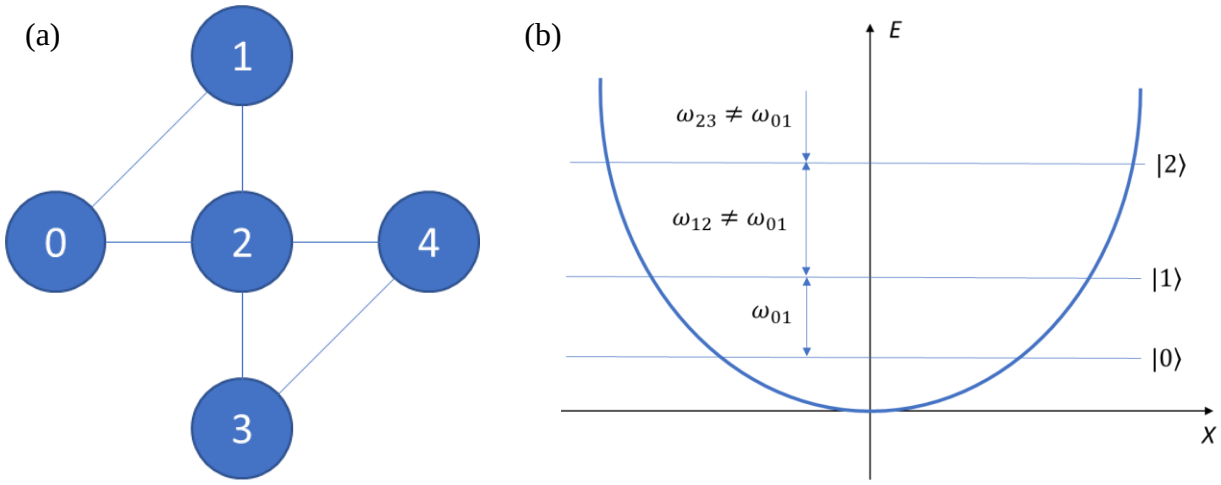

Figure 12: IBM quantum processor overview. (a) The qubits of the 5 qubit processors are arranged in a bow-tie pattern. The nodes are the josephson junction qubits and the edges between them indicate which qubits can be entangled. (b) The physics of the qubit can be modeled as a quantum anharmonic oscillator, where the irregular state is induced between states $|0\rangle$ and $|1\rangle$. This is accomplished using the nonlinear characteristics of the Josephson junction [31].

4.2 Quantum Perceptron Experimental Application

Here the 5 qubit Tenerife processor will be utilized to perform experiments of a two input QP architecture with two input qubits and one bias qubit as shown in Figure 5. Once again, U_i is the input state preparation for the input and bias states.

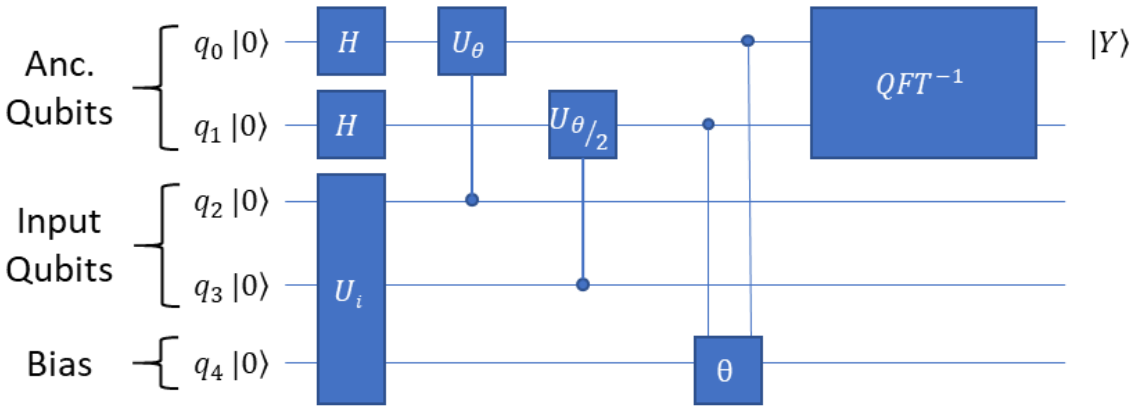

Figure 13: Experimental circuit configuration for QP. Reference figure 6 for an explanation of the nonstandard gate used to describe the bias weight gate.

$$U_i|\psi_i\rangle = |X_{in}\rangle \otimes |X_{bias}\rangle \tag{20}$$

And the nonstandard θ gate represents the dynamic entanglement of the bias qubit, q_4, with the ancillary qubits, q_0 and q_1.

As mentioned in the previous section, the QP will be run experimentally by plugging the final values from the results of the simulation into the gates constructing the algorithm on the IBM platform. The results will then be recorded and compared to the theoretical simulation results. Based on an existing training algorithm [37], I expect to see these results replicated in simulation and ultimately see theory translated to experiment. It is anticipated that the quantum processor will be able to replicate the general patterns produced by the simulations, however the probabilities will not be as strong due to noise in the hardware caused by effects such as

decoherence [21]. In an attempt to combat this noise, each experiment will be run 8192 times, the greatest amount IBM allows, and record the number of times a $|1\rangle$ is measured as the output. While this obviously does not rid the experiment of error entirely, it mitigates it by demonstrating statistically what state the system is in.

4.3 Training Pattern Experimental Results

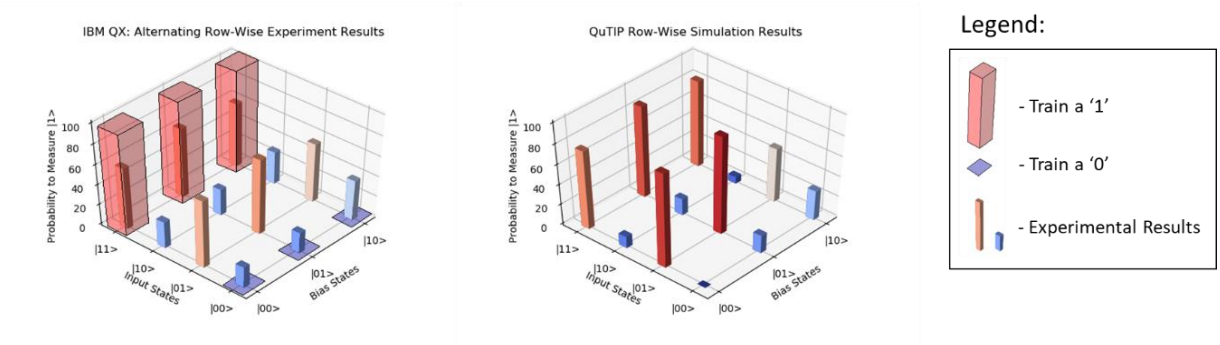

Figure 14: Experimental vs simulation results for input-dependent training pattern. The leftmost image displays the results obtained from the quantum processor and displays the six training values used to generate this system. The middle image the results from the simulation. While decoherence has neutered the extreme values expected by simulation, the general pattern is still well preserved.

The first example covered (Figure 14) is an alternating horizontal pattern achieved when the input neuron weight resolves to 1.68π radians and the bias neuron weight is 0.56π radians. This system output is primarily dependent on changes in input neuron weight and is largely independent of the bias neuron. As a result, this configuration does not afford the bias the opportunity to assist with calculations, and thus the null in, null out will be present in this

configuration. Otherwise stated, there is a low probability of measuring a $|1\rangle$ for all bias states when the input state is $|00\rangle$. The maximally segregated solution would have resulted in the input neuron weight settling at 2π and the bias neuron would have been disabled with a weight of 0. This would have led to all even state rows producing a 0% chance of measuring a state $|1\rangle$ and all odd state rows producing 100% chance of measuring $|1\rangle$ in simulation. Experimentally, this is closer to 25% and 75% respectively due to noise and decoherence effects that plague quantum processors. This is not necessarily the most ideal solution; it would simply represent the case in which the output is completely dependent on the input neuron and independent of the bias neuron. While it is not unreasonable to assume this was the desired output, it demonstrates the algorithm's ability to choose patterns with incremental amounts of dependency on each neuron. This experiment also demonstrates the QP's ability to resolve data with as much distance between the two groups as the output space will allow.

Table 2: Horizontal Pattern Weight Summary

Final Input Weight (Radians)	1.68π
Final Bias Weight (Radians)	0.56π

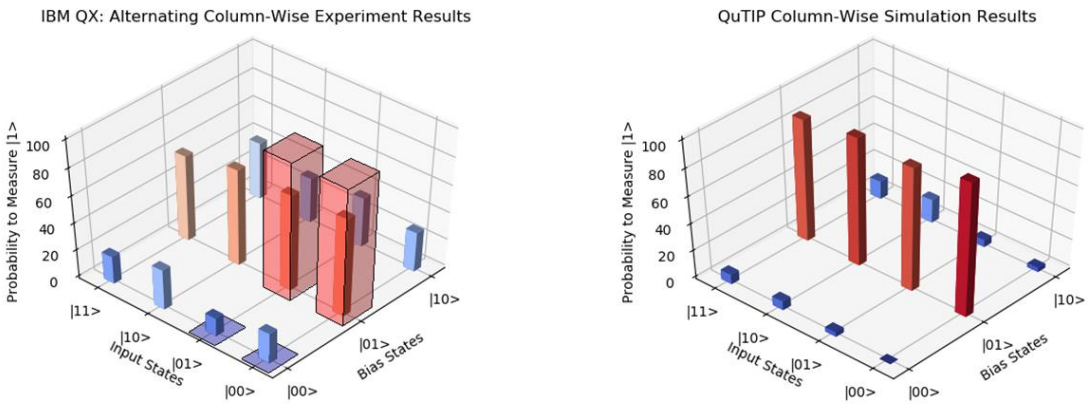

Figure 15: Experimental vs simulation results for bias-dependent training pattern.

The next pattern exhibited (Figure 15) is an alternating vertical pattern achieved when the input neuron weight resolves to 0.24π radians and the bias neuron weight is 2.16π radians. This is inverted from the previous case in that it is primarily dependent on the bias neuron and independent of the input neuron. The result is an output space where the even rows tend to a lower chance of measuring |1⟩ and the odd row tends to a higher chance of measuring |1⟩. This effectively gives the ability to switch the bias "on" by choosing bias state |01⟩ over |00⟩ or |10⟩, which will allow the output a high probability of measuring |1⟩ even when the input state is |00⟩. It also demonstrates that the QP is reasonably able to processes data points that are close together.

Table 3: Vertical Pattern Weight Summary

Final Input Weight (Radians)	0.24π
Final Bias Weight (Radians)	2.16π

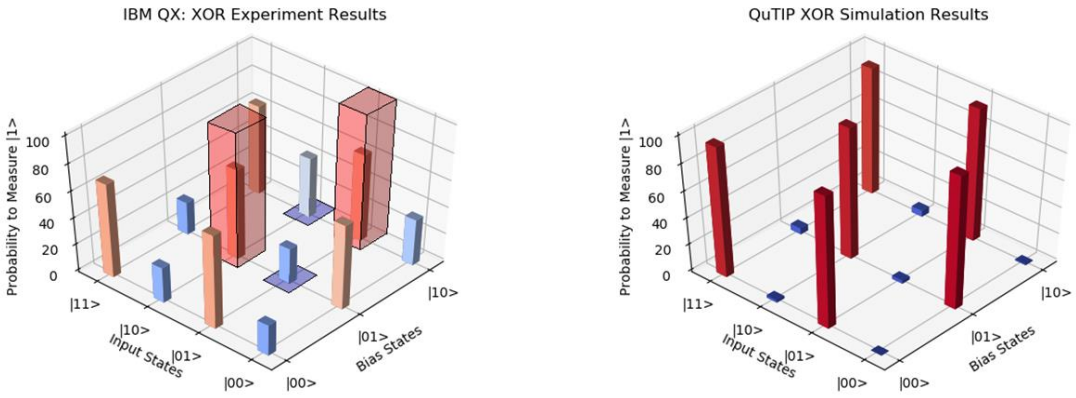

Figure 16: Experimental vs simulation results for XOR training pattern.

The next pattern exhibited is the ever-important XOR demonstration (Figure 16), accomplished by resolving the input neuron weight to 1.88π radians and the bias neuron weight is 1.92π radians. Presenting as a checkerboard pattern, this output space shows dependence on both neurons in order to generate this result, alternating between higher and lower output state percentage tendencies to measure a $|1\rangle$ as input states are iterated either row-wise or column-wise. This is a particularly important result as it illustrates a fundamental advantage that the QP has over its classical counterpart. This is a nonlinear data set and a classical perceptron would not be able to separate this data or create this output pattern.

Table 4: XOR Pattern Weight Summary

Final Input Weight (Radians)	1.88π
Final Bias Weight (Radians)	1.92π

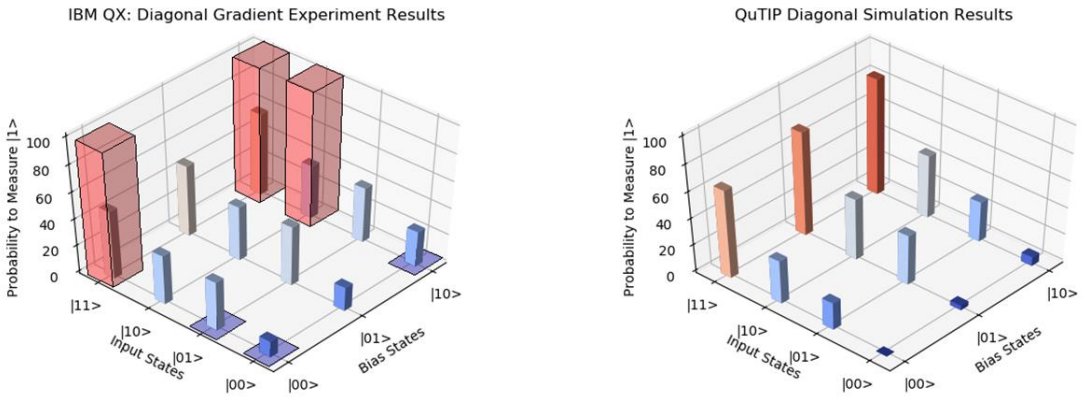

Figure 17: Experimental vs simulation results for diagonal gradient training pattern.

A more irregular pattern that shows a diagonal gradient (Figure 17), which is created when the input neuron weight settles to 3.40π radians and the bias neuron weigh to 3.76π radians. The primary purpose here is to show that a more organic pattern can be processed and extrapolated by the QP. Here, the QP has deduced that the output percentage rises primarily with the input state, but also has a slight positive association with the bias state as well. This creates a shape in output space with a strong vertical gradient and weak horizontal gradient, resulting in the overall diagonal gradient as previously mentioned. A notable feature of this experiment would be the middle two rows that match between the simulation and experiment. Output percentages close to either 100% or 0% calculated in simulation tend to be lowered or raised respectively in experiment due to noise as previously mentioned by as much as 25-30%. However, output states calculated in simulation that tend towards 50% do not seem to be dramatically different when replicated in experiment. This is most likely due to the fact that creation of these states already involves generating mostly noise, so the added noise from the environment does not affect it further.

Table 5: Diagonal Gradient Pattern Weight Summary

Final Input Weight (Radians)	3.40π
Final Bias Weight (Radians)	3.76π

These sample patterns shown here serve to demonstrate the ability of the QP to take in data and successfully generalize the information. In this context, it refers to the patterns generated in the output space as a result of the final weight combinations after training has been completed. While this is by no means an exhaustive case study, it sufficiently demonstrates the basic classification abilities of the QP for two input sources. Furthermore, it shows how using an input neuron as a bias signal not only allows the QP to output a high signal given a null input, but more generally its ability to extend the calculation capabilities of the QP as a whole.

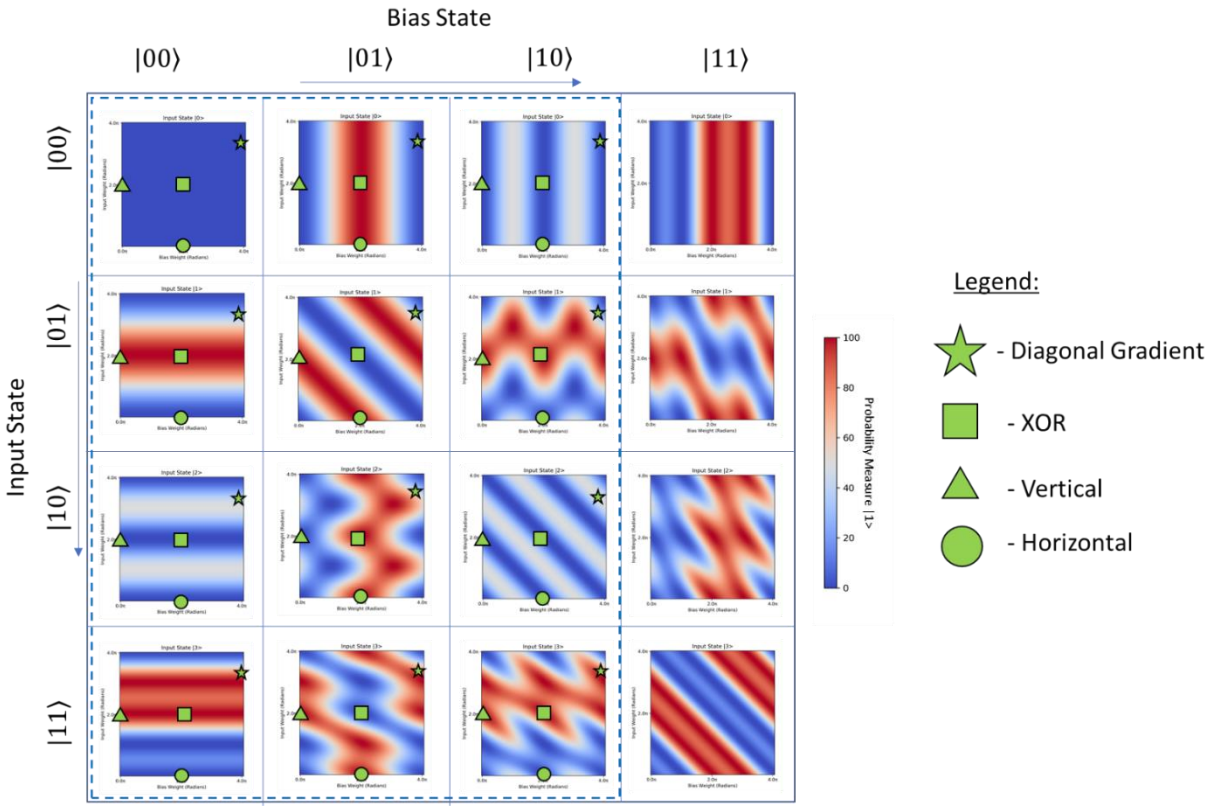

Figure 18: QI map tracking the weights of the experimental results. Different shaped markers correspond to different experiments. This demonstrates where the experiments ended up in output space and provides an example of how to use the QI maps.

4.4 Quantum Image Processing

As an example of a practical application of the QP, a basic architecture for image processing was devised. This was accomplished by leveraging the XOR configuration as demonstrated in the previous section, which will be created here by setting the weights of both neurons to 2π. The weights will be left static for this task and the bias state will switch between states |00⟩ and |01⟩ which will allow each pixel's output to be set to either 1 or 0. In this configuration, the signals to the input neuron tracks the location of the pixels of the image and

the bias neuron represents the pixel color value. As the QuTiP simulations create a noiseless environment for the images and therefor only had binary outputs to work with, only black and white images could be generated. However, decoherence in the IBM QX quantum processors will lead to percentages other than 100% or 0%, which will allow for the generation of grayscale images. Another obstacle to overcome the limitation of the output space of the QP being limited to four pixels, the images were simply processed four pixels at a time. The locations of these four-pixel subdivisions will be tracked by an array and, once the image has been fully processed by the quantum hardware, the complete image will be assembled.

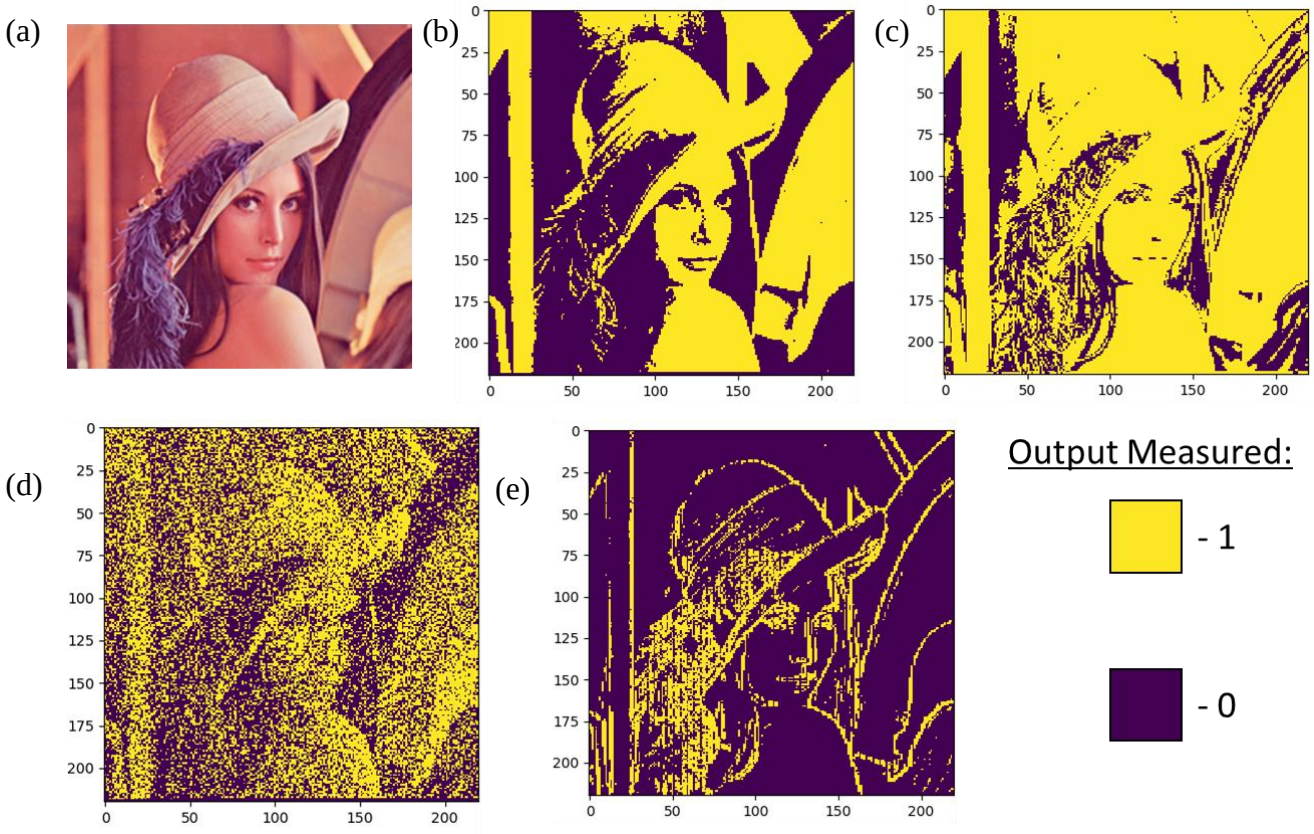

Figure 19: Basic scheme for quantum image processing using a QP. (a) The original image that will be processed by the algorithm. (b) High-pass filter with the cuttoff pixel value at 127.5 (c) Band-stop filter that filters out pixel values between 51 and 102 (d) Gaussian filter that uses a random number generator to determine the output state of the image. If the random number is higher than the pixel value, then set the state to high and vice versa. (e) Edge detection algorithm that checks if neighboring pixels are within a preset delta from the current pixel. If they are all similar, set the output to low and give a high output for differing pixel values. The delta here is ±15.

Displayed in the figure are examples of some basic image filters that the QP is capable of producing. These examples are all simulations and represent the results in a noiseless environment, which is why all pixel outputs measure as either 1 or 0. The original image, (a), is fed into the QP and the grayscale pixel values determine which state the bias will take, which in turn determines the final result of the system. This pixel value can be processed by a number of different conditions that lead to the filters demonstrated here. Starting with image (b), which is a simple high-pass filter where the cut-off is at the 50% mark. The grayscale pixels can take on values from 0 to 255, so this cutoff value is 127.5. This means that whenever a pixel from the image was fed into the QP, if its intensity value was greater than 127.5 then the bias state was set such that a high output would be expected and vice versa for a pixel value less than the cut-off. The final image is effectively produced from putting together the recorded output due to the determined bias states in accordance with these sensed pixel values and processing algorithms. The next instance is of a band-stop filter with a pixel value range of 51-102, which is clearly not a prevalent range in the image as most of the output space is still active. The third example is a Gaussian filter, which attempts to mimic grayscales using only black and white by statistically assigning a bias state based on the pixel intensity which should have an averaging effect over the area of many pixels. When the QP, receives the pixel value, a random number is also generated at the same time and compared to this value. If the random value is higher, then the bias is set to its high state, if it is lower, then it is set to low. The final example is that of a basic edge detection algorithm, in which the current pixel value is checked against the pixel values to the left and underneath and given a threshold value. If either surrounding pixel is determined to be sufficiently different, then the QP determines the current pixel to be an edge and sets the bias

state to high, otherwise it is set to low. In this particular example, the surrounding pixel values are checked against a delta of 15 from the pixel in question.

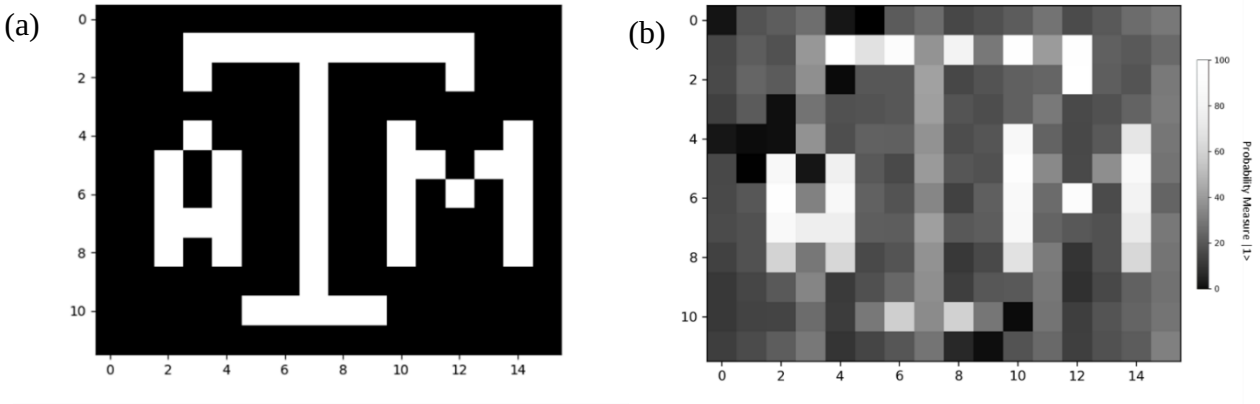

Figure 20: Experimental example of quantum image processing with QP.

Once the simulations confirmed that the QP architecture could successfully process images, the next step was to run an image on IBM's processors, this time using Qiskit as it is much more adept to the high-volume nature of this work. The high-pass filter is shown in the example accomplished by creating quantum circuits for each input state with the bias high and bias low state. The pixel value triggers which bias state is sent to the quantum processor and the results were recorded in an array. These experiments were run for 1024 repetitions in order to be more efficient with the amount of credits used, sacrificing some quality of the output in exchange for running more experiments at a time. Recording the counts of having measured state $|1\rangle$ in output has introduced a grayscale into what was previously an exclusively black and white image.

While it is clear that this technology is still proof of concept, it should offer advantages over classical processors as most image processing techniques are linear algebra based. Operations in quantum processors are based on linear algebra and lend themselves well to tasks such as this. For example, as the algorithm expands, the image will be able to take advantage of global memory which allows changes in one pixel to ripple throughout the whole image. Of course there is also the obvious limitation that the weights were artificially left static due to a current poor understanding of how to efficiently navigate quantum space. A single point in quantum space that sufficiently served the purposes of this experiment was chosen and the weights were locked, and this was enough to use the QP as a quantum graphics processing unit. However, this negates the QP's most notable feature, its ability to learn. With a more sophisticated understanding of the quantum output space, it would be possible to train the QP to recognize or replicate images. This exercise demonstrates the flexibility of the QP to operate as a learning vehicle as well as a processing unit.

CHAPTER V

CONCLUSION

Chapter 2 introduced a streamlined algorithm for a QP that is modular and can be easily expanded upon. Inspired by the classical perceptron, it is able to process input state information in combination with weight information from multiple sources which are convolved in order to produce an output. If active, the input signal will trigger the weight to perform a calculation which is stored as a phase revolution on a corresponding ancillary qubit. After a series of these revolutions have been performed on the ancillary qubits, an IQFT will be performed on these qubits to approximate the overall phase of the system.

Chapter 3 covered the QuTiP simulations of the QP providing a solid theoretical background showing the algorithm could be trained and generalize data. This also included QI maps which were created by iterating through all input states and weights and mapping the resulting output state. These maps were important in developing a working knowledge of the function of the algorithm and in making design choices when designing experiments on the IBM systems. Future work would include quasiprobability functions to classify and navigate QI space, such as Wigner functions.

Results from chapter 4 demonstrate that this QP can be implemented on current quantum hardware. The effects of noise become clear as the average error between theoretical results and experimental results is around 25-30%, however the overall patterns are generally very well preserved between the two cases. Additionally, an application for the QP was found in basic image processing. Demonstrated were several basic filters simulated on black and white images

followed by an experimental demonstrated in which a grayscale became possible due to noise in the processor.

REFERENCES

1. Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. "Quantum Speed-up for Unsupervised Learning." journal article. Machine Learning 90, no. 2 (February 01 2013): 261-87.
2. G. Aleksandrowicz, et al. "Qiskit: An Open-Source Framework for Quantum Computing." Accessed on: Mar 16 (2019).
3. Aradhyamath Poornima, Naghabhushana N. M., Rohitha Ujjinimatad. "Matrix Representation of Quantum Gates." International Journal of Computer Applications 159, 8 (2017): 6.
4. Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. "Quantum Machine Learning." Nature 549 (09/13/online 2017): 195.
5. Danilo Bruschi, Deborah Joseph, and Paul Young. "A Structural Overview of Np Optimization Problems." Paper presented at the Optimal Algorithms, Berlin, Heidelberg, 1989.
6. Courtland, Rachel. "Transistors Could Stop Shrinking in 2021." IEEE Spectrum, 2016, https://spectrum.ieee.org/semiconductors/devices/transistors-could-stop-shrinking-in-2021.
7. Cybenko, G. "Approximation by Superpositions of a Sigmoidal Function." journal article. Mathematics of Control, Signals and Systems 2, no. 4 (December 01 1989): 303-14.

8. Daskin, A. "A Simple Quantum Neural Net with a Periodic Activation Function." Paper presented at the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 7-10 Oct. 2018.
9. David Deutsch, and Richard Jozsa. "Rapid Solution of Problems by Quantum Computation." Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences 439, no. 1907 (1992): 553-58.
10. Feynman, Richard P. "Quantum Mechanical Computers." journal article. Foundations of Physics 16, no. 6 (June 01 1986): 507-31.
11. Feynman, Richard P. "Simulating Physics with Computers." journal article. International Journal of Theoretical Physics 21, no. 6 (June 01 1982): 467-88.
12. Grover, Lov K. "Quantum Computers Can Search Rapidly by Using Almost Any Transformation." Physical Review Letters 80, no. 19 (05/11/ 1998): 4329-32.
13. Grover, Lov K. "Quantum Mechanics Helps in Searching for a Needle in a Haystack." Physical Review Letters 79, no. 2 (07/14/ 1997): 325-28.
14. Johansson, J. R., P. D. Nation, and Franco Nori. "QuTiP: An Open-Source Python Framework for the Dynamics of Open Quantum Systems." Computer Physics Communications 183, no. 8 (2012/08/01/ 2012): 1760-72.
15. Jones, J. A. *Quantum Information, Computation and Communication*. 1 ed. Cambridge: Cambridge University Press, 2012.
16. Le Bellac, Michel. *A Short Introduction to Quantum Information and Quantum Computation*. Cambridge: Cambridge University Press, 2006.

17. U. Leonhardt, P.L. Knight, and A. Miller. *Measuring the Quantum State of Light.* Cambridge University Press, 1997.
18. John M. Martinis, S. Nam, J. Aumentado, and C. Urbina. "Rabi Oscillations in a Large Josephson-Junction Qubit." Physical Review Letters 89, no. 11 (08/21/ 2002): 117901.
19. Marvin Minsky, and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.
20. Ajit Narayanan, and Tammy Menneer. "Quantum Artificial Neural Network Architectures and Components." Information Sciences 128, no. 3 (2000/10/01/ 2000): 231-55.
21. Asier Ozaeta, and Peter L. McMahon. "Decoherence of up to 8-Qubit Entangled States in a 16-Qubit Superconducting Quantum Processor." Quantum Science and Technology 4, no. 2 (2019/04/17 2019): 025015.
22. Petrov, Christo. "Big Data Statistics 2019." techjury, 2019, https://techjury.net/stats-about/big-data-statistics/.
23. Pittenger, Arthur O. *An Introduction to Quantum Computing Algorithms*. Birkhauser Boston, Inc., 1999.
24. Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962.
25. R. P. Rundle, P. W. Mills, Todd Tilma, J. H. Samson, and M. J. Everitt. "Simple Procedure for Phase-Space Measurement and Entanglement Validation." Physical Review A 96, no. 2 (08/10/ 2017): 022117.

26. Schaller, R. R. "Moore's Law: Past, Present and Future." IEEE Spectrum 34, no. 6 (1997): 52-59.
27. Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. "The Quest for a Quantum Neural Network." journal article. Quantum Information Processing 13, no. 11 (November 01 2014): 2567-86.
28. Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione . "Simulating a Perceptron on a Quantum Computer." Physics Letters A 379, no. 7 (2015/03/20/ 2015): 660-63.
29. Shor, P. W. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring." Paper presented at the Proceedings 35th Annual Symposium on Foundations of Computer Science, 20-22 Nov. 1994.
30. Shor, P. W. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." SIAM Review 41, no. 2 (1999): 303-32.
31. M. Steffen, D. P. DiVincenzo, J. M. Chow, T. N. Theis, and M. B. Ketchen. "Quantum Computing: An Ibm Perspective." IBM Journal of Research and Development 55, no. 5 (2011): 13-11.
32. Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. "An Artificial Neuron Implemented on an Actual Quantum Processor." Nature Quantum Information 5, no. 1 (2019/03/29 2019): 26.
33. Martijn P. van den Heuvel, Cornelis J. Stam, René S. Kahn, and Hilleke E. Hulshoff Pol. "Efficiency of Functional Brain Networks and Intellectual Performance." The Journal of Neuroscience 29, no. 23 (2009): 7619-24.

34. Wigner, E. P. "On the Quantum Correction for Thermodynamic Equilibrium." In Part I: Physical Chemistry. Part Ii: Solid State Physics, edited by Arthur S. Wightman, 110-20. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997.

35. Williams, R. S. "What's Next? [the End of Moore's Law]." Computing in Science & Engineering 19, no. 2 (2017): 7-13.

36. Yu-Lin Wu, et al. "Fabrication of Al/Alox/Al Josephson Junctions and Superconducting Quantum Circuits by Shadow Evaporation and a Dynamic Oxidation Process." Chinese Physics B 22, no. 6 (2013/06 2013) : 060309.

37. Alexandre Y. Yamamoto, Kyle M. Sundqvist, Peng Li, and H. Rusty Harris. "Simulation of a Multidimensional Input Quantum Perceptron." journal article. Quantum Information Processing 17, no. 6 (April 19 2018): 128.

38. Jinming Zou, Yi Han, and Sung-Sau So. "Overview of Artificial Neural Networks." In Artificial Neural Networks: Methods and Applications, edited by David J. Livingstone, 14-22. Totowa, NJ: Humana Press, 2009.