NAVAL SURFACE WARFARE CENTER
PANAMA CITY DIVISION
PANAMA CITY, FL 32407-7001

TECHNICAL REPORT
NSWC PCD TR-2023-002

QUANTOM COMPUTING FOR MACHINE LEARNING: AN INTRODUCTION

PRINCIPAL CONTRIBUTORS:

Dominic Byrne
Advanced Signal Processing & Automatic Target Recognition Branch (X23)
Unmanned Systems, Automation & Processing Division (X20)
Science & Technology Department (X)

Matthew Cook
Advanced Signal Processing & Automatic Target Recognition Branch (X23)
Unmanned Systems, Automation & Processing Division (X20)
Science & Technology Department (X)

Ethan Evans
Automation & Dynamics Branch (X22)
Unmanned Systems, Automation & Processing Division (X20)
Science & Technology Department (X)

SEPTEMBER 2023

This Page Intentionally Left Blank

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (*DD-MM-YYYY*)	2. REPORT TYPE	3. DATES COVERED (From – To)
20-09-2023	Technical Report	1-10-2022 to 1-06-2023

4. Title of Technical Report
Quantum Computing for Machine Learning: An Introduction

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dominic Byrne, Matthew Cook, Ethan Evans

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

COMMANDING OFFICER
NAVAL SURFACE WARFARE CENTER, PANAMA CITY DIVISION
110 VERNON AVENUE
PANAMA CITY, FL 32407-7001

8. PERFORMING ORGANIZATION REPORT NUMBER
Tr-2023-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NISE
110 VERNON AVENUE
PANAMA CITY, FL 32407-7001

10. SPONSOR/MONITOR'S ACRONYM(S)
NISE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A: Approved for public release; distribution is unlimited. Administrative or operational use; March 2022. Other U.S. requests shall be referred to Naval Surface Warfare Center, Panama City Division (E13), Panama City, FL 32407-7001.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This paper provides an introduction to quantum machine learning, exploring the potential benefits of using quantum computing principles and algorithms that may improve upon classical machine learning approaches. Quantum computing utilizes particles governed by quantum mechanics for computational purposes, leveraging properties like superposition and entanglement for information representation and manipulation. Quantum machine learning applies these principles to enhance classical machine learning models, potentially reducing network size and training time on quantum hardware. The paper covers basic quantum mechanics principles, including superposition, phase space, and entanglement, and introduces the concept of quantum gates that exploit these properties. It also reviews classical deep learning concepts, such as artificial neural networks, gradient descent, and backpropagation, before delving into trainable quantum circuits as neural networks. An example problem demonstrates the potential advantages of quantum neural networks, and the appendices provide detailed derivations. The paper aims to help researchers new to quantum mechanics and machine learning develop their expertise more efficiently.

15. SUBJECT TERMS
Quantum Computing, Machine Learning

16. SECURITY CLASSIFICATION OF: Unclassified			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		68	MATTHEW COOK 19b. TELEPHONE NUMBER (Include area code) 850-771-8364

This Page Intentionally Left Blank

ABSTRACT

Quantum computing has quickly become a highly active research area, and quantum machine learning has emerged as a potential manifestation of classical machine learning on quantum hardware. The widespread successes of classical machine learning in classification problems are extremely attractive, however they come at the cost of an exponential growth of parameters in modern network architectures (e.g. Generative Pre-trained Transformers). A possible benefit in addressing such problems with quantum networks is an increased expressibility of quantum bits over classical bits, which through quantum machine learning leads to an increased expressibility of a quantum neuron.

Quantum computing is founded on the premise of using particles that are governed by quantum mechanics for the purposes of computation by leveraging key aspects such as superposition and entanglement. These properties have theoretical advantage in representing and manipulating information. Namely superposition allows for a fundamental bit of information to encode a continuous spectrum, while entanglement allows non-local effects to manipulate encoded information. Circuits of quantum gates are used to perform quantum computations, and when parameterized, can be optimized, or trained, using traditional methods in optimization. This leads to a quantum machine learning framework where classical information embedded in quantum bits can take advantage of quantum phenomena and increased expressibility for a potential reduction in network size and training time on quantum hardware.

This manuscript serves as introductory material for researchers that are new to the areas of quantum mechanics and machine learning, in order to decrease the timeframe needed for developing new expertise. The notion of a Turing machine is used as a foundation and motivation for creating computers out of quantum hardware. Next, basic principles and notation of quantum mechanics are introduced, including superposition, phase space on the Bloch sphere, and entanglement of multiple quantum bits. A basic review of classical digital logic is used to propose notions of quantum gates that may leverage these key properties by a universal set of quantum gates. Next, we introduce classical deep learning concepts such as the artificial neural network, the gradient descent algorithm and its stochastic generalization, and the standard backpropagation approach to training a neural network. These are used as a foundation for introducing trainable quantum circuits as neural networks, including a derivation of the analogous gradient descent approach and its generalizations and methods of encoding classical information in a quantum circuit. Finally, these topics are combined in an illustrative example problem that highlights a potential advantage of quantum neural networks. The accompanying appendices offer greater detail of various derivations that are provided throughout the manuscript.

SIGNATURE PAGE

This technical note has been prepared, reviewed and approved by the Advanced Signal Processing & Automatic Target Recognition Branch (X23) and the Automation & Dynamics Branch (X22).

MATTHEW COOK
Advanced Signal Processing & Automatic Target Recognition Branch (X23)
Unmanned Systems, Automation & Processing Division (X20)
Science & Technology Department (X)

DOMINIC BRYNE
Advanced Signal Processing & Automatic Target Recognition Branch (X23)
Unmanned Systems, Automation & Processing Division (X20)
Science & Technology Department (X)

ETHAN EVANS
Automation & Dynamics Branch (X22)
Unmanned Systems, Automation & Processing Division (X20)
Science & Technology Department (X)

DARSHAN BYNER
Head, Advanced Signal Processing & ATR (X23)

RICHARD TATUM
Head, Automation & Dynamics (X22)

DR. FRANK CROSBY
Head, Unmanned Systems Automation & Processing Department (X)

TABLE OF CONTENTS

Section	Page
WHY QUANTUM COMPUTING	1
QUANTUM COMPUTING OVERVIEW	1
Quantum Physics - Notation and Intro	1
Superposition and Entanglement	8
Fundamentals of Classical Computing	10
Fundamentals of Quantum Computing	11
Single-Qubit Gates	12
Two-Qubit Gates	15
The Quantum Universal Gate Set	16
Fundamental Challenges of Quantum Computing	17
INTRODUCTION TO CLASSICAL DEEP LEARNING ALGORITHMS	19
Overview of Classical Machine Learning Problems and Terminology	19
Neural Networks	21
Shared Weight Neural Networks	22
Gradient Descent	24
Backpropagation (Gradient descent for Neural Networks)	27
CURRENT STATE OF QUANTUM COMPUTING	28
Quantum Circuit Architecture	28
Embedding Classical Data in Quantum Circuits	31
Quantum Machine Learning	32
Parameter-Shift Rule	33
Stochastic Parameter-Shift Rule	37
Quantum Computing Example – The XOR Problem	40
CONCLUSIONS	46
REFERENCES	47
APPENDIX	51
A. Binary Dimensionality Reduction Example	51
B. Proof of the Pauli Commutator identity	52
C. Baker-Campbell-Hausdorff Derivation/Proof	53
D. Derivation of the Derivative of a Parametric Exponential Operator	55

FIGURES

Figure **Page**

Figure 1: Bloch sphere representation of a qubit 9
Figure 2: The AND gate 10
Figure 3: The OR gate 11
Figure 4: Perceptron Example 21
Figure 5: Multilayer Perceptron Example 22
Figure 6: Simple convolution example 23
Figure 7: Simple convolution layer 24
Figure 8: Example of gradient calculation. 25
Figure 9: The XOR gate 29
Figure 10: The quantum CNOT gate 30
Figure 11: Simple quantum circuit example 30
Figure 12: Example of block encoding method 32
Figure 13: Example quantum machine learning circuit 33
Figure 14: Circuits for parameter a1 upshift (top) and downshift (bottom) 36
Figure 15: Quantum circuit to solve XOR problem 40
Figure 16: Modified XOR circuit used in training 41
Figure 17: Generated XOR dataset 42
Figure 18: Loss by batch over the training period 43
Figure 19: Validation accuracy vs batch over training period 43
Figure 20: Correct (left) and network assigned (right) labels for test data 44
Figure 21: Theta parameter training 44
Figure 22: Alpha parameter training 45

TABLE OF ALGORITHM

Algorithm **Page**

Algorithm 1: Mini-batch gradient descent algorithm. 27
Algorithm 2: Process for quantum machine learning 37

WHY QUANTUM COMPUTING

The notion of a universal computer was first characterized by the description of the Turing machine [1] based on an infinitely long tape, or computation register, which through the machine could be used to encode any algorithm. These notions, along with the advent of the transistor, eventually led to the von Neumann architecture of computing we have today, where a central processing unit (CPU) performs sequential operations on an encoded stream of information, and separately stores this information in a memory register. With the trend of increasing transistor density per unit area, eventually a limit could be perceived. In addition, the polynomial time computational complexity constraint of this computing architecture, as suggested by the Church-Turing thesis, made the quest for alternative architectures inevitable [2].

The idea of leveraging quantum physics to perform computations was first suggested by Feynman, and is by no means the only alternative to the von Neumann architecture (see [3], [4], and [5]), however its appeal is based on the ability to leverage attractive properties such as superposition and entanglement, which allow for a unique framework. Herein, one must carefully construct algorithms which yield a choreography of operations that use sequential constructive and destructive interference to promote the correct solution in probability while effectively cancelling out the probability of incorrect solutions, all without a-priori knowledge of the correct solution. In principle, this approach promises an exponential speedup over the von Neumann architecture [6].

The novelty and unique challenges of leveraging quantum physics for computing has led to not just one, but numerous computing architectures explored by a growing contingent of companies and organizations, each seeking to establish their approach as the dominant approach, and to be the first to demonstrate quantum supremacy. Among these are: a) using superconducting qubits (often referred to as SQUIDs) by IBM, Google, USTC, and Rigetti, b) trapped ions by IonQ and others, c) photonics by Xanadu and others, and trapped Rydberg atoms by QuEra and others. Beyond these hardware implementations, are several other floating notions, such as building quantum computing on qudits (d-level quantum bits, e.g. ternary quantum bit) as opposed to the binary quantum bit (qubit) [7], as well as abandoning the discrete framework altogether and instead building computing on a continuum quantum state [8] [9].

Outside the goal of universal quantum computing are also specific quantum realizations for specific applications, as in the case of quantum annealing [10] [11] for optimization problems, and quantum machine learning [12] [13] for learning problems. The rapid and heavy investment in quantum computing has led to a so-called race for quantum supremacy in which proponents suggest it can be achieved as soon as the early 2030s [14], yet there remain substantial hurdles before this dream may be realized.

QUANTUM COMPUTING OVERVIEW

Quantum Physics - Notation and Intro

Quantum computing relies on the manipulation and measurement of quantum phenomena in order to process information. The behavior of quantum phenomena is described by quantum physics, so an understanding of quantum computing necessitates a basic understanding of

quantum physics. Quantum physics introduces math notation, known as Dirac notation, which is often quite foreign to other disciplines yet simplifies the introduction of main concepts and carries over to quantum computing. Dirac notation utilizes linear algebra, probability and statistics, as well as notational conventions used in physics.

Linear algebra is used in Dirac notation to describe quantum states as vectors, and to describe the physical processes that can impart change to a quantum state as matrices[1], which in quantum literature are referred to as operators. Operators 'operate' on these quantum states to give some new quantum state. A column vector is by convention used to represent some physical quantum state, for example whether an electron is in the spin up state or spin down state or a photon is in the horizontal or vertical linear polarization state. A vector state ψ (the variable ψ is often used in literature to describe a generic quantum state, usually called a 'wavefunction') with two quantities would normally be written as:

$$\vec{\psi} = \begin{bmatrix} a \\ b \end{bmatrix}. \tag{1}$$

In Dirac notation, the vector state ψ would be written as:

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix}, \tag{2}$$

where $|\cdot\rangle$ is the right side of a bracket $\langle \cdot | \cdot \rangle$ and is called a "ket" vector. The complex conjugate transpose of a vector state is also widely used, and is written as:

$$\vec{\psi}^{*\,T} = \begin{bmatrix} a^* & b^* \end{bmatrix} = \vec{\psi}^{\dagger}, \tag{3}$$

where * is the complex conjugate, T is the transpose, and the dagger (†) is a shorthand notation in quantum physics to combine the * and T into one symbol † for the complex conjugate transposed. In Dirac notation, the complex conjugate transposed of a vector state ψ is written as:

$$\langle\psi| = \begin{bmatrix} a^* & b^* \end{bmatrix}, \tag{4}$$

where $\langle \cdot |$ is the left side of a bracket $\langle \cdot | \cdot \rangle$ and is called a "bra" vector. A bra and a ket written next to each other in a full bracket denotes an inner product between the two vector states:

$$\langle\psi_1|\psi_2\rangle = \vec{\psi}_1^{\dagger} \cdot \vec{\psi}_2 = \begin{bmatrix} a_1^* & b_1^* \end{bmatrix} \cdot \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} = a_1^* a_2 + b_1^* b_2. \tag{5}$$

Unitary operators, represented by matrices, are used to change the quantum state. For example, some state ψ_i can be modified to another state, ψ_{i+1} after the application of an operator $\hat{U}$ (the

[1] In the more general setting, an operator can act on an infinite vector space, but this brief introduction limits vectors to finite dimensional vector spaces where operators are simply described by standard matrices.

hat $\widehat{\ }$ symbol is often used to denote an operator, though it is not always used). Operators are usually given capital letters for variable names, and a unitary operator is a special type of operator that satisfies the property:

$$\widehat{U}\widehat{U}^{\dagger} = \widehat{U}^{\dagger}\widehat{U} = \hat{I},$$

(6)

where $\hat{I}$ is the identity operator and corresponds to imparting *no change* to a quantum state. The action of an operator, represented as a matrix, is written as:

$$|\psi_{i+1}\rangle = \widehat{U}|\psi_i\rangle = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}\begin{bmatrix} a_i \\ b_i \end{bmatrix} = \begin{bmatrix} u_{11}a_i + u_{12}b_i \\ u_{21}a_i + u_{22}b_i \end{bmatrix} = \begin{bmatrix} a_{i+1} \\ b_{i+1} \end{bmatrix}.$$

(7)

One can similarly apply complex conjugates and transposes to an operator,

$$\widehat{U}^{\dagger} = \begin{bmatrix} u_{11}^* & u_{21}^* \\ u_{12}^* & u_{22}^* \end{bmatrix},$$

(8)

and the dagger of an operator is typically applied to bra vectors to update them:

$$\begin{aligned} \langle\psi_{i+1}| = \langle\psi_i|\widehat{U}^{\dagger} &= \begin{bmatrix} a_i^* & b_i^* \end{bmatrix}\begin{bmatrix} u_{11}^* & u_{21}^* \\ u_{12}^* & u_{22}^* \end{bmatrix} \\ &= \begin{bmatrix} a_i^*u_{11}^* + b_i^*u_{12}^* & a_i^*u_{21}^* + b_i^*u_{22}^* \end{bmatrix} \\ &= \begin{bmatrix} a_{i+1}^* & b_{i+1}^* \end{bmatrix}. \end{aligned}$$

(9)

Some operators are used to represent measurable properties of quantum systems and are called 'observables' or 'observable operators'. These operators satisfy a slightly weaker requirement of being Hermitian. Hermitian operators are defined by satisfying the following relation:

$$\hat{A} = \hat{A}^{\dagger}.$$

(10)

An important property of Hermitian operators is that one can easily show that they have all real eigenvalues despite having potentially all complex entries. Furthermore, Hermitian operators typically represent 'measurable' operators, and in such cases their eigenvalues correspond to familiar quantities such as position, momentum, energy, etc., which also must be real-valued.

The average value of all possible outcomes, based on the probability of each outcome, is referred to as the expectation value (or expected value) of that observable with respect to that quantum state. In Dirac notation, the expectation value is expressed as:

$$\langle\psi|\hat{A}|\psi\rangle,$$

(11)

where $\hat{A}$ is some observable (which we represent here by a matrix), and $|\psi\rangle$ is the quantum state just before the state is measured. The expectation value describes the average measurement outcome, so despite the discrete values of single measurements, an expectation value can (and

usually will) give a non-discrete value that does not correspond to a single possible measurement outcome. The possible discrete-valued outcomes of each measurement are given by the eigenvalues of observable $\hat{A}$.

Performing the calculation above will yield the average of the possible outcomes weighted by the probability of each outcome occurring. Expectation values can also be estimated empirically by performing repeated measurements and averaging the results. The probability will be inherent in the empirical histogram of each of the possible outcomes, and if measured an infinite number of times would result in the exact expectation value. Since infinite measurements are not practically feasible, an estimate for the expectation value can be obtained by a finite number of measurements and is often treated as the empirical expectation.

Another important calculation in quantum physics is the probability of some general quantum state $|\psi\rangle$ being in the specified, known state $|\phi\rangle$. This is calculated by the magnitude squared of the inner product between $|\psi\rangle$ and $|\phi\rangle$, which in Dirac notation is represented as:

$$|\langle\phi|\psi\rangle|^2 = (\langle\phi|\psi\rangle)^\dagger\langle\phi|\psi\rangle = \langle\psi|\phi\rangle\langle\phi|\psi\rangle. \quad (12)$$

The inner product $\langle\phi|\psi\rangle$ is the projection of the state $|\psi\rangle$ onto the desired measurement state $|\phi\rangle$. The inner product describes the 'amount of overlap' due to the projection, and is called the amplitude of the probability or the 'probability amplitude'. The square of the magnitude of the probability amplitude (which can also be computed by multiplication of the inner product with its complex conjugate), gives a real value that corresponds to the probability of finding the state $|\psi\rangle$ in the state $|\phi\rangle$ when measured.

In quantum computing, most calculations are performed in the computational basis formed by the zero state $|0\rangle$ and the one state $|1\rangle$ where:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (13)$$

These states are unit vectors because they have a norm (or magnitude, computed as the inner product of a state with itself) of one:

$$\langle 0|0\rangle = 1 \text{ and } \langle 1|1\rangle = 1. \quad (14)$$

They are also orthogonal to each other, meaning inner products with each other yield zero:

$$\langle 0|1\rangle = 0 \text{ and } \langle 1|0\rangle = 0 \quad (15)$$

Finally, the basis is 'complete', meaning that any arbitrary vector can be written as a linear combination of these two basis elements:

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} = a\begin{bmatrix} 1 \\ 0 \end{bmatrix} + b\begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (16)$$

Together, these three properties describe the basis as being a complete orthonormal basis.

So far these states seem similar to the binary zeros and ones that are used to represent values in classical computing, however quantum mechanics offers more possible states, which quantum computing seeks to utilize. While

$$|\psi\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |\psi\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(17)

would be the only possible states in a classical computing scheme, the state

$$|\psi\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

(18)

is a legitimate, though unnormalized, state in quantum mechanics. Normalization factors need to be added to ensure state probabilities sum to unity, which will be elucidated by an example. First, the importance of this state being allowed should be highlighted. The state $|\psi\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is a linear combination of the zero state and the one state:

$$|\psi\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |0\rangle + |1\rangle,$$

(19)

and represents some quantum state at a certain point or time before measurement. The fact that multiple possible states are present simultaneously is called superposition, and is a property unique to quantum mechanics. This superposition of states is itself a quantum state; operations and gates applied to the state $|\psi\rangle$ are applied in their normal manner and act on both parts of the state present in the superposition state simultaneously. This key quantum phenomenon leads to a larger computational space that quantum computing seeks to leverage; the classical binary representation is replaced with a continuous space of possible superposition states.

However, once a measurement is made only one of the possible measurable values remains, which was predicted with some probability. So before measurement, both quantum states exist simultaneously in the quantum superposition state, but upon measurement this quantum state "collapses", and only one basis state is observed while the information for the other state is lost. This probability is mathematically determined by the complex coefficients that multiply each basis element, which are also used to normalize the quantum state.

Now to show why those coefficients are needed, a counter example is presented. Starting with the unnormalized superposition state (without coefficients) introduced earlier:

$$|\psi\rangle = |0\rangle + |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

(20)

If we take its norm

$$\langle\psi|\psi\rangle = \begin{bmatrix} 1^* & 1^* \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2,$$

(21)

we obtain a value of 2. The issue with this becomes apparent when the magnitude of the inner product is squared:

$$|\langle\psi|\psi\rangle|^2 = (\langle\psi|\psi\rangle)^\dagger\langle\psi|\psi\rangle = \langle\psi|\psi\rangle\langle\psi|\psi\rangle = 2 * 2 = 4.$$

(22)

As described earlier, the magnitude of the inner product of two states is squared gives the probability that one state will be in the other state when measured. If the same operation is performed on a state with itself, it yields the probability of the state being in its own state, which must yield a probability of 1 or 100%. Without any normalization coefficients, this calculation gives a probability of 4, or 400%, which is a nonsensical and nonphysical result. Therefore, a condition is imposed that the magnitude of the norm of a state squared must always give a probability of one to ensure the physics remains consistent and logical:

$$|\langle\psi|\psi\rangle|^2 \equiv 1.$$

(23)

This condition also implies that

$$|\langle\psi|\psi\rangle|^2 = (\langle\psi|\psi\rangle)^\dagger\langle\psi|\psi\rangle = \langle\psi|\psi\rangle\langle\psi|\psi\rangle = (\langle\psi|\psi\rangle)^2 \equiv 1$$

(24)

$$\langle\psi|\psi\rangle \equiv 1.$$

Coefficients are added to a state to ensure this condition remains true. This is often referred to as normalization. For the above superposition state:

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} = a|0\rangle + b|1\rangle = a\begin{bmatrix} 1 \\ 0 \end{bmatrix} + b\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(25)

this condition gives constraints on a and b:

$$\langle\psi|\psi\rangle = \begin{bmatrix} a^* & b^* \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = a^*a + b^*b \equiv 1$$

$$|\langle\psi|\psi\rangle|^2 = \begin{bmatrix} a^* & b^* \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} a^* & b^* \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = a^*aa^*a + a^*ab^*b + b^*ba^*a + b^*bb^*b \equiv 1$$

(26)

These constraints are the same, as squaring the first yields the second.

The coefficients a and b are also used to describe the probability of measuring the state associated with that coefficient. For example, say a state $|\psi\rangle$ has coefficients $a = 1/\sqrt{3}$ and $b = \sqrt{2/3}$:

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

(27)

Then to calculate the probability of measuring the zero state from this superposition state, take the square of the magnitude of the inner product of this state with the zero state:

$$|\langle 0|\psi\rangle|^2 = \begin{bmatrix} \frac{1}{\sqrt{3}} & \sqrt{\frac{2}{3}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \sqrt{\frac{2}{3}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{3}} & \sqrt{\frac{2}{3}} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \sqrt{\frac{2}{3}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & \sqrt{\frac{2}{3}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} = \frac{1}{3}.$$

(28)

This means there is a $1/3$ probability of measuring the eigenvalue associated with the "zero" state. Performing a similar calculation for the one state yields $2/3$, meaning there is a $2/3$ probability of measuring the eigenvalue associated with the one state.

It is also important to note that the magnitude squared of a and b gives the respective probabilities of each state's eigenvalues being measured:

$$a^*a = \frac{1}{3} \quad and \quad b^*b = \frac{2}{3}$$

(29)

This can be seen clearly by doing the probability calculations for measuring the zero state and the one state with a generic state $|\psi\rangle = a|0\rangle + b|1\rangle$ where a and b are left as variables.

Zero state:

$$|\langle 0|\psi\rangle|^2 = \begin{bmatrix} a^* & b^* \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

$$= \begin{bmatrix} a^* & b^* \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a^* & b^* \end{bmatrix} \begin{bmatrix} a \\ 0 \end{bmatrix} = a^*a$$

(30)

One state:

$$|\langle 1|\psi\rangle|^2 = \begin{bmatrix} a^* & b^* \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

$$= \begin{bmatrix} a^* & b^* \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a^* & b^* \end{bmatrix} \begin{bmatrix} 0 \\ b \end{bmatrix} = b^*b$$

(31)

For further reading, see references [15], [16], [17], [18], [19].

Superposition and Entanglement

The two fundamental properties of quantum physics that are leveraged by quantum computing are superposition and entanglement. When operating with qubits, oftentimes the pertinent physical property being leveraged for quantum computing is known as the spin state. Let $|s\rangle$ represent the spin state of a 2-state particle (i.e. spin-up and spin-down). Then superposition is simply an expression of the spin state in terms of its basis elements

$$|s\rangle = \alpha|up\rangle + \beta|down\rangle$$

(32)

Where $\alpha, \beta \in \mathbb{C}$ are complex numbers such that the state is normalized, i.e. $|\alpha|^2 + |\beta|^2 = 1$, $|up\rangle$ represents the particle being spin-up, and $|down\rangle$ represents the particle being spin-down. At a given time, the particle is in a linear combination of its two basis states, which is also referred to as being in a superposition of its basis states, as described earlier. Upon measurement, the superposition *collapses* to a single state outcome, which is often referred to as 'wave function collapse'.

While this notion is quite simple from the perspective of linear algebra, it is far more interesting from the perspective of probability theory. As explored earlier, these complex coefficients of superposition are closely related to the respective probabilities of each outcome. For this reason, often times they are referred to as *probability amplitudes*. The association to probabilities in the classical sense comes by squaring these amplitudes, namely

$$P(|up\rangle) = |\alpha|^2, \qquad P(|down\rangle) = |\beta|^2.$$

(33)

Thus, if for the moment, we take these amplitudes to be purely real, assign the $|up\rangle$ state to be a 2-vector of unit length pointing in a positive-z direction, and assign the $|down\rangle$ state to be a 2-vector of unit length pointing in a negative-z direction, then superposition describes a 2D circle of unit length since we require $|\alpha|^2 + |\beta|^2 = 1$. The vector orthogonal to the z-axis in this circle corresponds to a state $|s\rangle = \frac{1}{\sqrt{2}}|up\rangle + \frac{1}{\sqrt{2}}|down\rangle$, where each outcome has equal probability, and this direction is assigned positive-x. Now, returning to the more general case of complex probability amplitudes, the imaginary direction creates a new y axis, and our 2D circular representation becomes a 3D sphere of unit length. This representation is called the Bloch sphere representation, and is depicted in Figure 1[2].

[2] This figure originated from https://demonstrations.wolfram.com/QubitsOnThePoincareBlochSphere/

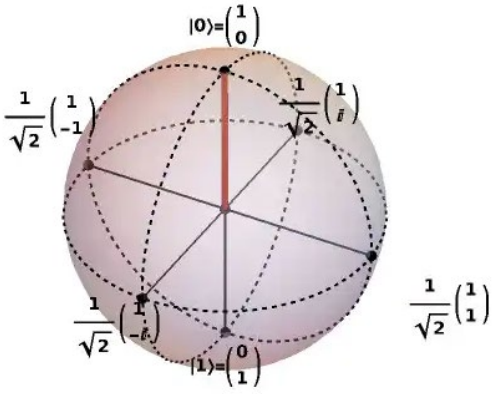


Figure 1: Bloch sphere representation of a qubit

The property of entanglement is more complex, but also emerges mathematically in a deceptively simple form. Essentially, if one has two 2-state spin particles

$$|s_1\rangle = \frac{1}{\sqrt{2}}|up\rangle + \frac{1}{\sqrt{2}}|down\rangle, \qquad |s_2\rangle = \frac{1}{\sqrt{2}}|down\rangle + \frac{1}{\sqrt{2}}|up\rangle,$$

(34)

and can express their combined state $|s_1, s_2\rangle$ as

$$|s_1, s_2\rangle = \frac{1}{\sqrt{2}}|up, down\rangle + \frac{1}{\sqrt{2}}|down, up\rangle,$$

(35)

then the two particles are said to be in a state of maximal entanglement. That is, since the total two-particle state is merely expressed by these two two-particle basis states, their states are coupled in such a way that if one were to measure the state of one of the particles (say in the up state), then they would have complete information of the state of the other (which must be in the down state). We say the particles are *maximally* entangled when each outcome has equal probability (50% for this example). For a two qubit system, there are four such maximally entangled states, known as Bell states. Since the act of measurement of one particle gives complete information of the other, entanglement is a nonlocal phenomenon, so that virtually all manipulations (i.e. through computing) of the state of one of the particles has some effect on the other particle, regardless of how separated the two particles are.

It is however not the case that all multi-particle systems are entangled. If one can separate the total system state into products of subsystem states, then the state is not entangled. This can be depicted in the following way. Say we have the two-particle system state:

$$|s_1, s_2\rangle = \alpha|up, down\rangle + \beta|up, up\rangle,$$

(36)

This system is not in an entangled state because it can be expressed as a product of subsystem states as:

$$|s_1, s_2\rangle = |up\rangle(\alpha|down\rangle + \beta|up\rangle).$$

(37)

Thus, if we were to measure the first particle, which must be in an up state, we would not gain any information about the state of the second particle, which could still be either in the up or the down state.

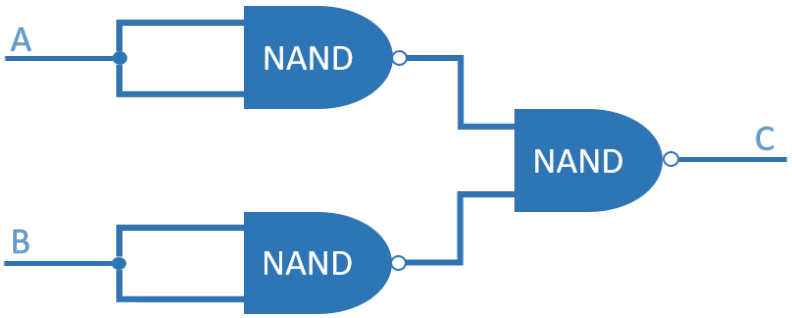
Fundamentals of Classical Computing

In classical computing, we construct every computation from a fundamental set of building blocks: logical gates. These gates act on pairs of bits by conditionally flipping, summing, or otherwise manipulating the bit pair to yield a single output bit. A simple example is the AND gate shown in Figure 2. The AND gate returns 1 if both A and B are 1, and returns 0 otherwise, which is analogous to multiplying the inputs. The opposite of the AND gate is the NAND (NOT AND) gate, which returns 1 if both A and B are 0, and 0 otherwise. In contrast, the OR gate returns 1 if either A or B is 1, and 0 otherwise, which is analogous to addition. The NOR (NOT OR) gate is its opposite, returning 1 if either A or B is 0, and 1 otherwise. Another important gate is the XOR gate, which returns 1 if A and B are different, and 0 if they are the same, analogous to an equality test. Its opposite, the XNOR gate, returns 0 if A and B are different, and returns 1 if they are the same.

Input		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2: The AND gate

These 6 gates may be applied in a sequence, referred to as a circuit, to create complex networks of logical operations that in turn may be used to construct any algorithm or set of operations. Connecting back to the ideas of Turing, these logical gate building blocks form a universal computer, and are the foundation of modern digital computing. An interesting thing to note is that many of these gates can be created by circuits of a different gate. For example, as depicted in Figure 3, one can construct an OR gate from a circuit of only NAND gates. Actually, one can construct *any* gate with a circuit of only NAND gates. Similarly, one can construct *any* gate with a circuit of only NOR gates. We refer to each of these individually as their own universal gate basis set. This is an important notion that will be explored shortly.



Input		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Figure 3: The OR gate

Fundamentals of Quantum Computing

Similar to the notion of a classical gate, the main thread of quantum computing research seeks to compose quantum computers out of similarly defined logic gates and circuits of logic gates in order to rebuild the classical computing architecture we are familiar with today from the ground-up, and thus produce a universal computer out of quantum components. Instead of acting on bits, these components act on quantum bits, or qubits, which are binary representations of quantum states in a 2-state system.

Consider again the spin state of an electron. This state can either be *spin-up* or *spin-down*. We previously used the ket notation to describe this as the $|up\rangle$ state and the $|down\rangle$ state, but equivalently, one can write them, respectively, as $|0\rangle$ and $|1\rangle$, so that we have encoded the two possible spin state outcomes into a binary representation. These encoded binary representations both simplify our notation, but also generalize to cases where the primary leveraged property is *not* spin, but some other 2-state property (e.g. the excited/ground state of a Rydberg atom).

The fundamental difference in how this binary state behaves is again the notion of superposition. When each qubit is observed (measured), it must be either in the $|0\rangle$ or the $|1\rangle$ state, however during the sequence of gate operations, it is described by a superposition state

$$|s\rangle = \alpha|0\rangle + \beta|1\rangle\,.$$

(38)

Thus, similarly, our quantum logic gates must be able to act not only on binary states, but also on *any* superposition of binary states. Since we always describe states with respect to this basis, it is convenient to write the state vector only in terms of the probability amplitudes α and β as

$$|s\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix},$$

(39)

Where by convention the top entry corresponds to the amplitude of the $|0\rangle$ basis state and the bottom entry corresponds to the amplitude of the $|1\rangle$ basis state.

Another complication arises from the time-reversibility of quantum mechanics. This requirement states that one must be able to run any sequence of operations *either* forwards *or* backwards. Mathematically, this means that every gate operation must be *one-to-one*, that is, an operation on some unique input produces some unique output. Thus each quantum gate operation must be *invertible*. This is not true of classical gates; a variety of different inputs can produce the same output, and furthermore, the size of the input space and the size of the output space need not match. This can be clearly seen in the AND logic gate diagram. Two inputs produce a single output, and the output is not unique to the set of inputs (e.g. there are several input pairs that produce 0). Thus, since we have a binary representation, and our input and output sizes must match, it is natural to describe quantum gate operations on single qubits using 2x2 matrices, and quantum gate operations on pairs of qubits using 4x4 matrices.

Finally, operations must preserve the normalization of the quantum state. As was explained earlier, quantum states, and thus gates that operate on quantum states, must preserve probability. Mathematically, this means that for an operator $\hat{A}$ acting on a state $|\psi\rangle$ as $\hat{A}\,|\psi\rangle = |\psi'\rangle$, the new state $|\psi'\rangle$ must satisfy $|\psi'|^2 = 1$. In the context of operator theory, this requirement implies that gate operations must be *unitary* and thus satisfy

$$\hat{A}\hat{A}^{\dagger} = \hat{A}\hat{A}^{-1} = \hat{I}. \tag{40}$$

Since we regard quantum gates as square matrices, we thus require unitary matrices. Notice that this condition is stronger than the invertibility condition, and that *any* unitary operator is also invertible. As such, typically these two requirements are just expressed as a single unitary requirement, despite originating from different fundamental concepts.

Single-Qubit Gates

Building on this intuition, we can now attempt to create quantum interpretations of classical logic gates, now represented as unitary matrices. The first gate we will attempt to reproduce is the NOT gate. Assume we have the general single-qubit state as before

$$|s\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}, \quad |\alpha|^2 + |\beta|^2 = 1. \tag{41}$$

Now, the NOT gate, often denoted as an operator by $\hat{X}$, should transform a $|0\rangle$ state to a $|1\rangle$ state and vice-versa, so that the amplitudes are flipped

$$\hat{X}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\beta\\ \alpha\end{bmatrix}. \tag{42}$$

It may be straightforward to see that the only matrix representation of $\hat{X}$ that satisfies this is

$$\hat{X} = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, \tag{43}$$

which is indeed the quantum NOT gate for a single qubit.

Before we move forward with two-qubit gates, it is interesting to note that while the only non-trivial single classical bit gate is the NOT gate, this is not the case for qubits. There are many non-trivial single-qubit gates, but two important single-qubit gates are the Z gate and the Hadamard gate. The single qubit Z gate is given by

$$\hat{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

(44)

This gate simply flips the sign of the amplitude of the $|1\rangle$ state while leaving the $|0\rangle$ state unchanged. Note that this does not change the probability of the $|1\rangle$ state, since probabilities are squares of amplitudes; instead, it adds a phase of π.

The Hadamard gate is given by

$$\hat{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

(45)

To understand the effect of this gate, consider the effect of applying $\hat{H}$ to the $|0\rangle$ state:

$$|s\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \hat{H}|s\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$

(46)

Similarly, consider the effect of applying $\hat{H}$ to the $|1\rangle$ state:

$$|s\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \hat{H}|s\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

(47)

These states are 'halfway' between the two basis states, thus the Hadamard gate generates superpositions where each basis state has a probability of ½. We may think of these two states as a *uniform* (symmetric and antisymmetric, resp.) superpositions, since the probabilities are uniformly distributed over outcomes (i.e. basis states). Note that $\hat{H}^2 = \hat{I}$, so applying the Hadamard gate to a uniform symmetric superposition returns the $|0\rangle$ state, and applying the Hadamard gate to a uniform antisymmetric superposition returns the $|1\rangle$ state. The Hadamard gate is typically used in circuits at the very beginning to generate a uniform superposition state that will be leveraged by the rest of the circuit.

We briefly mention a few other single qubit gates. The X and Z gates described above are often described as Pauli gates, of which there are three. The Pauli Y gate is given by

$$\hat{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}.$$

(48)

A well-known feature of Pauli operators is that the set $\{I, X, Y, Z\}$ forms a basis over the space of 2x2 complex Hermitian operators. The phase gate S is given by

$$\hat{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix},$$

(49)

and is given this name simply because it maps a generic state $|s\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ to the state $|s\rangle = \begin{bmatrix} \alpha \\ i\beta \end{bmatrix}$, thus creating a complex phase.

The $\pi/8$ gate is given the symbol T and is given by

$$\hat{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}.$$

(50)

Note that $\sqrt{i} = e^{\frac{i\pi}{4}} = \cos\left(\frac{\pi}{4}\right) + i\,sin\left(\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}$, so the T gate is the square root of the S gate and corresponds to a $\frac{\pi}{4}$ rotation in the complex plane as opposed to a $\frac{\pi}{2}$ rotation with the S gate.

Finally, the rotation gates $R_x(\theta),\ R_y(\theta),\ R_z(\theta)$ are given by

$$\hat{R}_X(\theta) = e^{-i\frac{\theta}{2}\cdot\hat{X}} = \begin{bmatrix} \cos\theta/2 & -i\sin\theta/2 \\ -i\sin\theta/2 & \cos\theta/2 \end{bmatrix}$$

$$\hat{R}_Y(\theta) = e^{-i\frac{\theta}{2}\cdot\hat{Y}} = \begin{bmatrix} \cos\theta/2 & -\sin\theta/2 \\ \sin\theta/2 & \cos\theta/2 \end{bmatrix}$$

$$\hat{R}_Z(\theta) = e^{-i\frac{\theta}{2}\cdot\hat{Z}} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}.$$

(51)

These gates are analogous to the rotation matrices in three Cartesian axes, and equivalently rotate the amplitude vector on the Bloch sphere described in Figure 1 about the corresponding axis. In this context of rotations on the Bloch sphere, one can generalize all single-qubit gates to a product of rotations. Namely, *any* arbitrary single-qubit gate U can be decomposed as

$$\hat{U} = e^{i\alpha}\hat{R}_z(\beta)\hat{R}_y(\gamma)\hat{R}_z(\delta) = e^{i\alpha}\begin{bmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{bmatrix}\begin{bmatrix} \cos\gamma/2 & -\sin\gamma/2 \\ \sin\gamma/2 & \cos\gamma/2 \end{bmatrix}\begin{bmatrix} e^{-i\delta/2} & 0 \\ 0 & e^{i\delta/2} \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma/2\, e^{i(\alpha-\beta/2-\delta/2)} & -\sin\gamma/2\, e^{i(\alpha-\beta/2+\delta/2)} \\ \sin\gamma/2\, e^{i(\alpha+\beta/2-\delta/2)} & \cos\gamma/2\, e^{i(\alpha+\beta/2+\delta/2)} \end{bmatrix},$$

(52)

which equates to four parameters on three sequential one-parameter gates (and a phase scaling parameter). This property is quite useful, and may be leveraged for the design of quantum circuits.

Two-Qubit Gates

In the case of two-qubits we have four outcomes, as opposed to the two in the single-qubit case. These are given by $|00\rangle, |01\rangle, |10\rangle$, and $|11\rangle$. Thus these states define the computational basis, and one can describe superpositions of these basis states in the general form

$$|s\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{bmatrix}\alpha_{00}\\ \alpha_{01}\\ \alpha_{10}\\ \alpha_{11}\end{bmatrix}. \quad (53)$$

Instead of measuring the entire state as in the case of a single qubit, here we may measure just one of the two qubits, which has a 'back action' effect on the other. Say we measure the first qubit. The probability of a $|0\rangle$ state on the first qubit is $|\alpha_{00}|^2 + |\alpha_{01}|^2$, and such an outcome would cause a post-measurement state of

$$|s'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} = \frac{1}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}}\begin{bmatrix}\alpha_{00}\\ \alpha_{01}\\ 0\\ 0\end{bmatrix} \quad (54)$$

Thus, two-qubit gates are defined by 4x4 unitary matrices. Among the most famous two-qubit gates is the CNOT gate, given by

$$\text{CNOT} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} = \begin{bmatrix}\hat{I} & \hat{0}\\ \hat{0} & \hat{X}\end{bmatrix}. \quad (55)$$

In essence, this gate flips the second qubit if the first qubit contains the $|1\rangle$ state, and does nothing otherwise. To see this, let us apply it to the $|00\rangle$ state:

$$\text{CNOT}\begin{bmatrix}1\\ 0\\ 0\\ 0\end{bmatrix} = \begin{bmatrix}1\\ 0\\ 0\\ 0\end{bmatrix} \quad (56)$$

Thus when the first qubit is $|0\rangle$, the total state is unchanged. The same occurs to the state $|01\rangle$. Next apply CNOT to the $|10\rangle$ state

$$\text{CNOT}|10\rangle = \text{CNOT}\begin{bmatrix}0\\ 0\\ 1\\ 0\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\\ 1\end{bmatrix} = |11\rangle, \quad (57)$$

So, the second qubit is flipped when the first qubit contains the $|1\rangle$ state. The same qubit flip occurs in the case of the $|11\rangle$ state, producing $|10\rangle$. More generally,

$$\text{CNOT}\begin{bmatrix}\alpha\\ \beta\\ \gamma\\ \delta\end{bmatrix} = \begin{bmatrix}\alpha\\ \beta\\ \delta\\ \gamma\end{bmatrix}, \quad \forall\, \alpha,\beta,\gamma,\delta \in \mathbb{C}\ \ s.t.\ \ |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1.$$

(58)

There are a variety of other two-qubit gates that use the first qubit as a control, and conditionally apply any of the single-qubit gates described above. Examples include the controlled-Z gate (sometimes called CZ), given by

$$\text{controlled-Z} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & -1\end{bmatrix},$$

(59)

and the controlled-phase gate (sometimes called CS), given by

$$\text{controlled-phase} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & i\end{bmatrix}.$$

(60)

Another widely used two-qubit gate is the swap gate, given by

$$\text{swap} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

(61)

This gate swaps the $|01\rangle$ state for the $|10\rangle$ state, but leaves other states unaffected. For the generic two-qubit state, we have

$$\text{swap}\begin{bmatrix}\alpha\\ \beta\\ \gamma\\ \delta\end{bmatrix} = \begin{bmatrix}\alpha\\ \gamma\\ \beta\\ \delta\end{bmatrix}, \quad \forall\, \alpha,\beta,\gamma,\delta \in \mathbb{C}\ \ s.t.\ \ |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1.$$

(62)

The Quantum Universal Gate Set

One can quickly see that the richness of the quantum description yields many more usable operations in comparison to the classical case; one has a variety of non-trivial single-qubit gates in contrast to only one non-trivial single bit gate. The basic intuition behind this feature is that quantum gates can cause complex amplitudes to *interfere* with each other, thus canceling out quantum amplitudes. The feature of having a much larger (uncountably infinite) gate set is much more pronounced with larger multi-qubit operators. The trade-off is that there is significant added complexity to define the universal gate basis set, that is, the set of quantum gates that can produce *any* unitary operator with sufficient precision.

The challenge with a universal quantum gate set is that it asks to compose a finite set of operators that when put in sequence can produce an uncountably infinite set of n-qubit operators. Thus, the claim of classical universality is weakened for the quantum case by only requiring that we produce any unitary operator with *sufficient precision*. Some requirements for a universal quantum gate set are that it must be able to *create* superposition (e.g. Hadamard), that it must be able to *create* entanglement, and that it must be able to create complex amplitudes as well as real ones.

It has been shown [6] that one such universal quantum gate set is the set {CNOT, S, $R_X(\pi/4)$}. This set is not unique; one can substitute the $R_X(\pi/4)$ gate for nearly any rotation gate. Such a universal set enables the quantum computing paradigm to recreate, and *generalize* the classical computing paradigm.

Fundamental Challenges of Quantum Computing

Here we describe one of the major difficulties in realizing quantum computers. The increased richness in the expressibility of a quantum superposition state also leads to one of the hardest problems in realizing quantum computing hardware, the problem of quantum error correction [18]. To elucidate this, consider first the classical computing case.

There is always noise that can induce errors in any information processing channel. As a result, even in classical computing there can sometimes be random errors that change a bit from a 0 to a 1, which are called bit flip errors. In classical computing, any channel of communicating information typically appends redundant information to a binary string in order that one can use the redundant information to decode the binary string and recover the intended information *despite* the error. For example a repetition code with majority voting is a simple classical error correction scheme, where if you intend to communicate a 0 bit, you instead send 000, and similarly to send the 1 bit you send 111. At the receiving end, you simply decode the bit of information based on the majority of 0 or 1 bits communicated, such that you can always protect against a single bit flip error.

In the quantum error correction case, since our qubit states are superpositions of basis states, errors can be understood as changing the probability amplitudes of outcomes. Consider a single qubit as before:

$$|s\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}, \quad |\alpha|^2 + |\beta|^2 = 1.$$

(63)

As in the case of the X gate, the qubit flip yields:

$$|s'\rangle = \begin{bmatrix}\beta\\ \alpha\end{bmatrix} = \beta|0\rangle + \alpha|1\rangle.$$

(64)

We can similarly apply the classical repetition code for qubits as:

$$|0\rangle \rightarrow |0\rangle_L \equiv |000\rangle,$$
$$|1\rangle \rightarrow |1\rangle_L \equiv |111\rangle,$$

(65)

where $|\cdot\rangle_L$ corresponds to a logical qubit, so that the single qubit state becomes a three-qubit state:

$$|s\rangle = \alpha|0\rangle_{\mathrm{L}} + \beta|1\rangle_L.$$

(66)

As before, we can diagnose and correct a single qubit flip by majority voting.

When viewed as a Pauli rotation on the Bloch sphere, a single bit flip is equivalent to a π rotation about the x-axis. A fundamental challenge arises upon realizing that the error may rotate our qubit not only about the x-axis, but about *any* axis in 3D. Thus for a single qubit, we must also perform a similar error diagnosis on the *phase* flip of a qubit, which has a similar flavor to the bit flip diagnosis, but in a rotated basis corresponding to the x-axis on the Bloch sphere

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$
$$|-\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

(67)

Logical qubits can be formed in this basis such that a phase flip is diagnosed using a similar majority voting repetition code.

A famous result in quantum error correction is known as the Shor code, which can correct an *arbitrary* error (i.e. any single qubit rotation error), by encoding a 9-qubit logical qubit, however there are codes that can correct any single qubit error by encoding as small as a 5-qubit logical qubit. Thus a quantum computer with only single qubit errors must implement 5 times the number of qubits to attain a given number of logical qubits.

Coupled to this issue is that circuits of increasing numbers of qubits suffer from *cross-talk* errors, which is when qubits in different channels become undesirably entangled to each other, which requires another level of quantum error correction. There are a variety of quantum errors that appear in hardware that must be diagnosed and corrected, which leads to a complex problem of quantum error correction over arbitrary width quantum circuits. A major challenge in quantum computing is finding codes that reduce the necessary number of qubits to achieve a logical qubit, and building a large enough quantum computer with enough fault tolerance to be able to perform useful operations on these logical qubits.

INTRODUCTION TO CLASSICAL DEEP LEARNING ALGORITHMS

The primary goal of machine learning and more specifically deep learning is to optimally approximate some functional mapping that encodes a challenging task. These tasks historically drew heavy inspiration from the everyday tasks that are accomplished by the animal brain. Today, a vast variety of interesting problems are addressed using machine learning and deep learning, from understanding protein folding for better drug discovery [20], to predicting complex weather patterns [21], to processing and/or translating language [22], to autonomous driving [23], to realizing nuclear fusion technology for green energy production [24].

For example when a fox sees the movement of an animal in the distance, its visual cortex must quickly determine if that animal is a bear and the fox should scurry away, or if the animal is a rabbit and the fox should pursue it. This general perception task is highly studied in machine learning literature and known as the task of classification. Here, the natural processes of the brain that classify the visual image of an animal as a bear or a rabbit can be understood as a functional mapping between images and classes of animals.

Another example is based on the motor cortex, where a human might want to pick up a glass of water in order to drink from it. The motor cortex evaluates the current position, velocity, and acceleration of the arm, and the current position of the cup, and must determine the electrical signals that are sent to the arm to cause it to extend and grasp the cup. This general task is also highly studied in robotics literature and known as the task of control (i.e. controlling the hand and arm to grasp the cup). Here again, the natural processes of the brain can be understood as a functional mapping between generalized locations and muscular actuation signals.

This sort of input-output mapping representation of a process or relationship is extremely flexible and general, perhaps universal. Using this framework, the goal of machine learning is to closely approximate the inherent relationship between input and output by a function that contains many, often millions, of flexible parameters often referred to as weights. The mapping is most commonly represented, or modeled, using a highly parameterized Artificial Neural Network (ANN), and the learning in machine learning refers to optimizing the parameters of the model so that it can mimic the process or relationship to high accuracy and precision. The adaptability and expressibility of ANNs have been one of the biggest motivators of their wide spread use. In this section, the basics of ANNs will be covered.

Overview of Classical Machine Learning Problems and Terminology

In machine learning the general goal is to learn a model from a system with known inputs, and sometimes outputs, so that the model can predict or enhance understanding of the underlying data. In the most general case there are two types of machine learning algorithms; supervised and unsupervised. In the unsupervised case, only the inputs to a system are known and the goal is generally to better describe the data itself. For example, a common unsupervised learning task is called clustering, wherein the machine learning model is attempting to find patterns in the data that identify common characteristics of portions of the data so that homogeneous data is grouped, or clustered, together. In the supervised case, both the inputs and outputs of the system are known and the goal is to accurately predict the systems outputs given the inputs. We will focus on the supervised case as that is more related to our current work.

In the supervised machine learning framework there are again two tasks that machine learning performs; classification and regression. In the classification task machine learning models seek

to identify membership of the input data to categories known as classes. For example, determining if an image contains a cat or a dog is understood as determining if the object in the image belongs to the 'cat' class or the 'dog' class. This type of model will often be referred to as a classifier, as its purpose is generally to assign the correct class to an input, where the outputs are effectively binary 'yes or 'no' for class membership of each class. The regression task is very similar except that instead of being binary in the output, models generally output a continuous value, e.g. given an image of a dog, predict the weight in kilograms of said dog. The differences in these tasks usually comes down to the type of data available and the setup of the optimization problem.

Once the task and model have been defined, the model's parameters are optimized such that the model fits the data as closely as possible. This optimization procedure is often referred to as training, and is the critical phase of a machine learning algorithm. Here, some function $\mathcal{F}$ that quantifies some abstract measure of distance, or error, between the predicted labels $\hat{y}$ and the true labels y of the training data x is minimized as

$$\theta_{\text{best}} = \arg\min_{\theta} \mathcal{F}(\hat{y}(\theta, x), y), \tag{68}$$

where $\mathcal{F}(\hat{y}(\theta, x), y)$ is the function to be minimized (e.g. mean squared error) measuring some notion of 'badness' of the model's prediction of the class label. This function is sometimes called an objective function, a cost function, or a loss function depending on the community. All machine learning models can be represented by an equation that maps the input to the desired output, and in this case the predicted labels $\hat{y}$ are a function of the model parameters θ and the training data x.

Before beginning to train, the available data is usually divided into three categories: training data, validation data, and test data. The training data is the data that is used to optimize the model parameters, while the validation data is used to prevent overfitting. Overfitting occurs when the optimization process learns the training data too well and does not generalize to other data. During training, the validation data is evaluated using the same function as the training data, generally referred to as the validation loss, however the model is never updated using the validation loss. The validation loss is monitored over training iterations, and overfitting is generally indicated by an increase in the validation loss for a decreasing training loss. When the validation loss begins increasing, training can be stopped to prevent overfitting. The final step in the machine learning process is the testing phase. Here the model is evaluated to determine true performance. This phase requires a completely blind set of data called test data, i.e. neither the validation nor training data can be used. This ensures that the performance of the model on the test data captures what typical performance would look like in practice.

If we consider a general classification task such as the one described above, the goal of training is to separate the input space into multiple regions where each region contains only a single class label. This clearly requires that the data is separable in some way; and the simplest case is where the data is linearly separable. Let's assume that our data has two classes, if a straight line can be drawn between the classes that data is linearly separable and any machine learning model will be able to correctly classify this data. This very rarely happens in real world data, thus many advanced models have been developed to handle data of various complexities, such as data that is not linearly separable. One such model will be covered next.

Neural Networks

The beginnings of ANNs trace back to 1957 with Rosenblatt's perceptron [25]. The perceptron classifier is a simple linear classifier/regression model. In the perceptron a group of weights are used to transform the input variables into the desired output. The perceptron uses the equation, $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{x}$ is the input vector, $\boldsymbol{W}$ are the weights, and $\boldsymbol{y}$ is the output which is either a scalar value or a new vector. This is a linear mapping which is severely limited in its real world application as demonstrated by the following example.

The XOR problem is often used for illustrations on the decision boundaries obtained with neural networks. The XOR problem generally consists of four clusters of data which are grouped into two classes. The clusters are generally created in a square, i.e. the four cluster centers are located at (-1, -1), (-1, 1), (1, 1), and (1, -1), with the diagonal clusters labeled as one class and the off-diagonal clusters as the opposite class. The perceptron is shown graphically on the left side of Figure 4, and the image on the right shows the limitations of the perceptron in solving the XOR problem. In the image, the different colors each indicate a different class. Since the perceptron is a linear classifier it can only define a simple linear decision boundary. Therefore, the perceptron is unable to correctly separate the diagonal and off-diagonal classes.

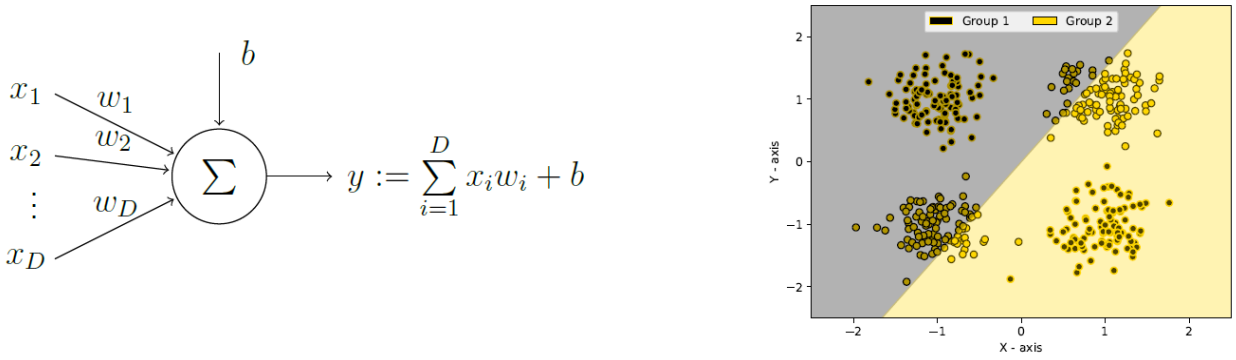


Figure 4: Perceptron Example

To improve upon the perceptron it was found that performance could be greatly improved by combining multiple perceptrons together. The combinatorial process involves creating layers of perceptrons. A layer is generated by having several perceptrons in parallel, with each perceptron using the same inputs. When combined in this fashion each perceptron is generally called a neuron. As the single perceptron case is simply an inner product between the inputs and the weights, a layer can be seen as a linear mapping from the input space to the space defined by the weights of each neuron. Thus each layer is computed as a matrix multiplication. The output from each linear mapping is then fed into a non-linear function often referred to as an activation function, which is named after its approximation of the step activation of a biological neuron. This is often referred to as the Wiener method [26], which has the non-linear activation following the summation in Figure 4. The purpose of the non-linear activation is to increase expressibility of the network, as without non-linearities a series of linear mappings will always reduce to a single linear mapping. Stacking multiple layers together, by connecting the outputs of a layer to the inputs of the following layer, creates what is called the Multi-Layered Perceptron (MLP) [27], and is shown in the left side of Figure 5. In the diagram each circle is considered a neuron. The outputs from the final layer, generally called the logits, are often converted to probabilities via the softmax function, which is defined as

$$\sigma(\boldsymbol{z})_i = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_j}} \text{ for } i = 1, \ldots, K,$$

(69)

where K is the number of logits in the output layer, $\boldsymbol{z}$ is the vector of output logits with z_i being the i^{th} element of the vector, and β is referred to as the temperature which is a hyperparameter that controls the smoothness of the softmax function.

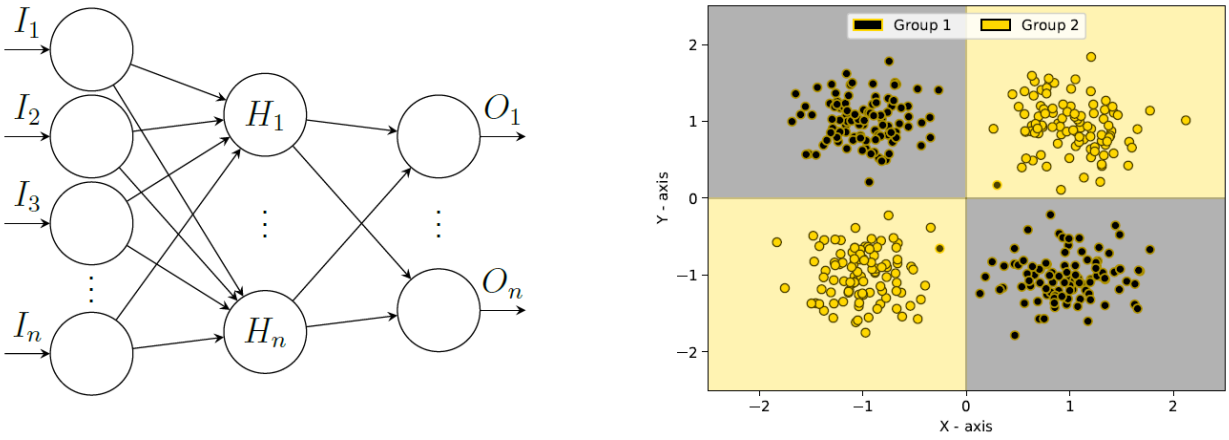


Figure 5: Multilayer Perceptron Example

With the addition of multiple layers and non-linearities the MLP is now capable of classifying data that is not linearly separated as in the right side of Figure 5. The stacking of layers allows the MLP to define multiple regions that can be separated linearly by the final classification layer. For example, in the XOR problem shown in the right side of Figure 5 the hidden layer can divide the feature space into the four quadrants shown. The classification layer then classifies the quadrants into their proper classes as the first layer projects the data into a space where the quadrants are linearly separable. To be considered a MLP there must be at least three layers: the input layer, the output layer, then at least one hidden layer. The hidden layer(s) fall between the other two layers as shown in the left side of Figure 5. In general multi-layer networks, there are many layers of matrix-vector multiplications which can be expressed as

$$f(\boldsymbol{x}_n; \boldsymbol{W}) = \sigma_{N_h}\left(\boldsymbol{W}_{N_h}\sigma_{N_{h-1}}\left(\boldsymbol{W}_{N_{h-1}}\sigma_{N_{h-2}}\left(\ldots\sigma_1(\boldsymbol{W}_1\boldsymbol{x}_n)\right)\right)\right),$$

(70)

where $\boldsymbol{x}_n$ is an input, N_H is the number of hidden layers, $\sigma_h(.), h = 1, 2, \ldots, N_H$ are activation functions, and $\boldsymbol{W}$ are the weights. To describe the potential capabilities of the MLP the Universal Approximation Theorem was proved in [28] which states that MLPs are capable of approximating any continuous function to an arbitrary accuracy given that the hidden layer is of sufficient size.

Shared Weight Neural Networks

The standard MLP is effective for a broad category of tasks, however the MLP also introduces a bias toward interconnectivity of every data point. While this is often a good strategy, it can introduce significant redundancy, and specifically many classification problems rely on images,

for which local relationships are more important. In adding more capability to the MLP, the next big advancement was the shared weight network from [29]. This is easiest to visualize via the convolutional neural network (CNN) [30]. Convolution networks are based on the standard convolution operation defined as:

$$(f * g_\theta)[n] = \sum_{m=0}^{N-1} f[m] g_\theta[n-m],$$

(71)

where f and g_θ are functions to be convolved, and g_θ is parameterized by θ and is often referred to as the convolution filter. Note that convolution is usually implemented as correlation in convolution networks, because correlation requires fewer operations and the convolution filter is learned therefore the two are effectively equivalent. To better explain what is happening see Figure 6. In the figure a simple convolution example is shown using two vectors, one of size 4 and another of size 3.

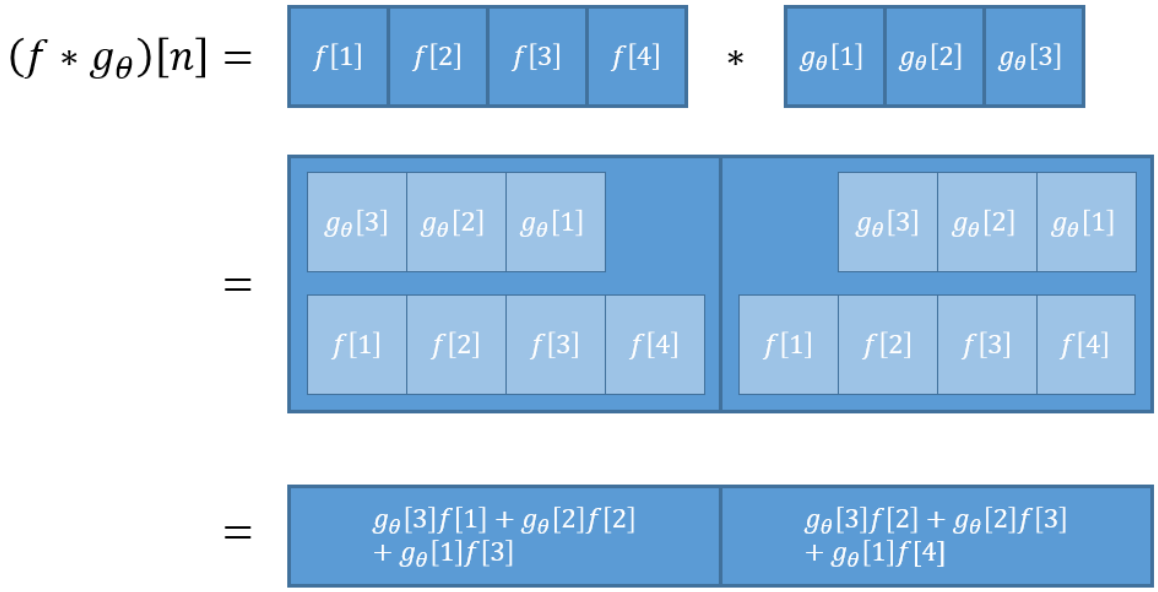


Figure 6: Simple convolution example

For this demo only the valid portion of the convolution is used meaning only the locations where two vectors fully overlap are used, these two positions are shown in the center line of the figure. The last line of the diagram shows the output of the convolution as an equation of the individual elements of f and g_θ. The last line can also be represented as a matrix multiplication as in:

$$(f * g_\theta)[n] = \begin{bmatrix} g_\theta[3] & g_\theta[2] & g_\theta[1] & 0 \\ 0 & g_\theta[3] & g_\theta[2] & g_\theta[1] \end{bmatrix} \begin{bmatrix} f[1] \\ f[2] \\ f[3] \\ f[4] \end{bmatrix}.$$

(72)

With this representation if we write the 2×4 matrix as $\boldsymbol{W}$, with $w_n = g_\theta[n]$, and the vector of $f[n]$ as $\boldsymbol{x}$, with $x_n = f[n]$ we are left with exactly the equation of a perceptron, $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$ as was seen before. With this new representation we can rewrite the convolution operation we started with as the simple single layer neural network shown in Figure 7.

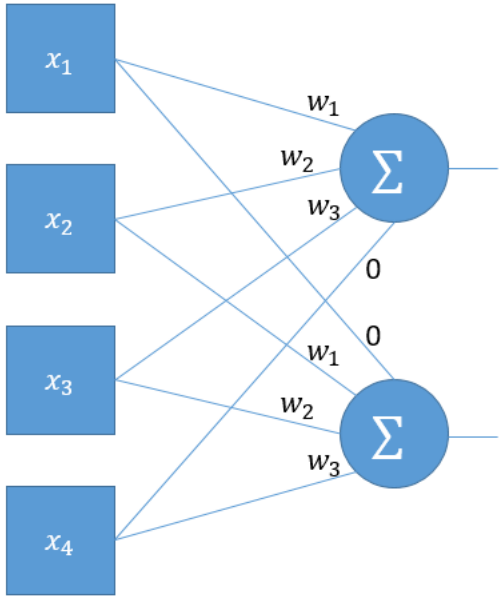


Figure 7: Simple convolution layer

In this representation the weights (w_1, w_2, w_3) are shared across both of the neurons represented by the summation symbols, thus a small set of weights can be shared across the input space. This type of network is in general applied to images, and can dramatically reduce the number of network parameters, which can reduce overfitting, reduce training time, and reduce model complexity.

Gradient Descent

Training for all types of neural networks uses some flavor of gradient descent. Gradient descent is an optimization strategy that is widely used for fitting many different types of models and data. Assuming a convex function, for a given point the sign of the gradient points away from the minimum. For example, in Figure 8, assume we are trying to find the minimum of the function $F(x)$ while starting at the point $x_1 = 2$. Computing the gradient of $F(x)$ at x_1 results in $F'(x_1) = 4$. If $F'(x_1)$ is added to x_1 we would move in the opposite direction of the actual minimum located at $x = 0$. This also happens for x_2 on the opposite side of the minimum. With this example we can see that by taking a small step in the direction of the negative gradient we can move closer to the true minimum function value.

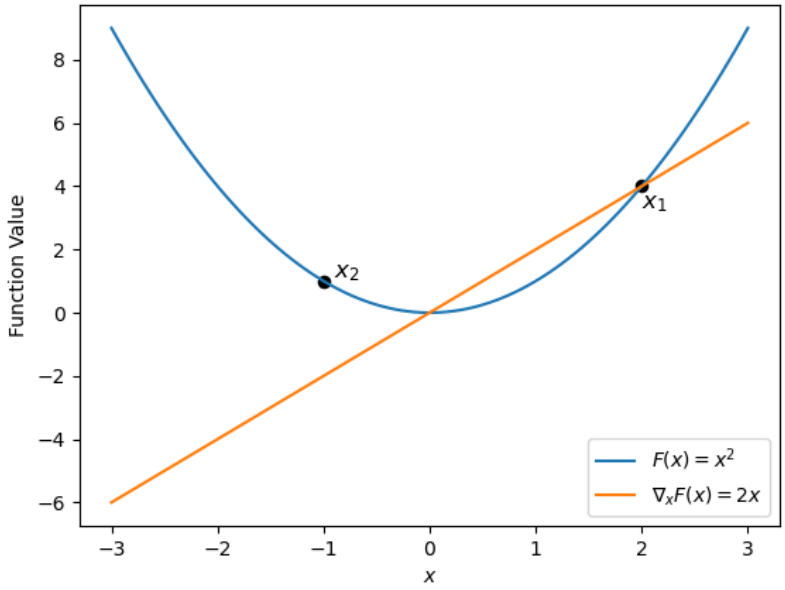


Figure 8: Example of gradient calculation.

Gradient descent is simply repeatedly over many iterations (update evaluations) computing the gradient and taking a small step in the direction of the gradient. Therefore, the rule for updating any model via gradient descent is as follows,

$$\boldsymbol{x}_{new} = \boldsymbol{x}_{old} - \lambda \nabla_{\boldsymbol{x}} F(\boldsymbol{x}),$$

(73)

where $F(\boldsymbol{x})$ is the objective function parameterized by the parameter $\boldsymbol{x}$ and λ is the step size that controls how far to move in the direction of the gradient. Repeatedly applying (73) until convergence will return the value of $\boldsymbol{x}$ that minimizes $F(\boldsymbol{x})$.

Gradient descent for machine learning follows the same basic formula as above except that the function being minimized typically takes the form,

$$\mathcal{F}(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\boldsymbol{W}, \boldsymbol{X}_i, \boldsymbol{y}_i),$$

(74)

where $\mathcal{L}(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$ is a function often called the loss function (squared error for example), which depends on the model parameters $\boldsymbol{w}$, training data $\boldsymbol{x}$, and training labels $\boldsymbol{y}$. The loss function is then summed over the n samples available in the training dataset, this is an unbiased estimator for the expected value of $\mathcal{L}(\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{y}_i)$ over the inputs $\boldsymbol{x}_i$. For example, using the perceptron discussed earlier as the model with mean squared error as the loss function yields,

$$\mathcal{F}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)^2.$$

(75)

Here the training data $\boldsymbol{X}$ and the training labels $\boldsymbol{y}$ are known, therefore optimization is done solely on the model parameters $\boldsymbol{w}$. Evaluating the gradient and using Equation (73) generates the following update equation for the perceptron algorithm,

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \frac{\lambda}{n}\left(\sum_{i=1}^{n}\left(y_i - \boldsymbol{w}_{old}^{\mathrm{T}}\boldsymbol{x}_i\right)\right)\boldsymbol{x}_i.$$

(76)

This method is commonly referred to as the batch gradient descent update. Here the entirety of the training set is used to compute a single update to the model, so there is only a single update in each iteration.

Another version of gradient descent is commonly called stochastic gradient descent. In this variant instead of using the entire training dataset to compute a single gradient update, only a single data point is used for each update. Therefore, the stochastic gradient descent update equation is reduced to,

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \lambda\left(y_i - \boldsymbol{w}_{old}^{\mathrm{T}}\boldsymbol{x}_i\right)\boldsymbol{x}_i.$$

(77)

The biggest difference between the two methods is in how many updates are performed. For the batch gradient descent there was only one update for each pass of the dataset, however, for the stochastic gradient descent method there will be n updates each pass. A pass through the dataset is commonly called an epoch. A way to interpret the differences between the batch and the stochastic versions is that, effectively, the batch method computes the average of all the individual updates to compute its one update. This allows a smoother convergence to the correct solution. The stochastic update leads to a much noisier convergence curve as the model is reacting to each data sample individually. The noisy convergence curve cab be beneficial though, as the randomness in the updates allows the model to jump out of local minima during training to potentially find better solutions.

There is a third type of gradient descent that is far more widely used, especially in the deep learning community, which is the mini-batch method. Mini-batch gradient descent is a hybrid of the batch and stochastic methods. Whereas the batch gradient method uses the entire dataset to compute the gradient, the mini-batch method uses only a small subset, larger than 1, to compute the gradient. This also allows for updating the model multiple times during an epoch but does not require updating after every single training sample. It also incorporates stochasticity by randomly sampling the mini-batches of data. The full algorithm for the mini-batch gradient descent is shown in Algorithm 1 below. This method incorporates the benefits of both methods by allowing the optimizer to have some randomness in the updates like the stochastic method but limits the amount of randomness by controlling the batch size. A small batch size increases the randomness, and thus optimization behaves more like the true stochastic version, while increasing the batch size reduces the randomness so the optimization behaves like the standard batch gradient descent method.

Algorithm 1: Mini-batch gradient descent algorithm.

Inputs: Training data $\boldsymbol{X}$, training labels $\boldsymbol{y}$, number of epochs N, batch size m

1. Randomly initialize model parameters $\boldsymbol{w}$
2. For $\boldsymbol{i} = \boldsymbol{1}, \dots, \boldsymbol{N}$ do
   - 2.1. Randomly shuffle $\boldsymbol{X}$ & $\boldsymbol{y}$
   - 2.2. Divide $\boldsymbol{X}$ & $\boldsymbol{y}$ into batches of size m ($\boldsymbol{X_j}$ & $\boldsymbol{y_j}$), save number of batches $\boldsymbol{M}$
   - 2.3. For $\boldsymbol{j} = \boldsymbol{1}, \dots, \boldsymbol{M}$ do
     - 2.3.1. $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{\lambda} \frac{\boldsymbol{1}}{\boldsymbol{2m}} \sum_{\boldsymbol{j}}^{\boldsymbol{m}} \boldsymbol{\nabla_w \mathcal{L}(w, X_j, y_j)}$
   - 2.4. If Converged
     - 2.4.1. Exit loop
3. Return $\boldsymbol{w}$

Backpropagation (Gradient descent for Neural Networks)

Gradient descent as it is defined above works well in many cases, however with neural networks the sheer number of parameters and serial-ness of the operations can make differentiating with respect to each parameter inefficient. Combining Equation (70) with the mean squared loss (for simplicity) leaves:

$$\mathcal{L}(\boldsymbol{x}_n, y_n, \boldsymbol{W}) = \frac{1}{2}\left(y_n - \sigma_{N_h}\left(\boldsymbol{W}_{N_h}\sigma_{N_{h-1}}\left(\boldsymbol{W}_{N_{h-1}}\sigma_{N_{h-2}}\left(\dots\sigma_1(\boldsymbol{W}_1\boldsymbol{x}_n)\right)\right)\right)\right)^2 \quad (78)$$

as the full loss function to be optimized. Following the process described above for gradient descent we would differentiate with respect to $\boldsymbol{W}$ to find the update equation. However, a single expression cannot be found to optimize over each individual $\boldsymbol{W}_{N_h}$ simultaneously as was done before. A separate update function could be found by differentiating with respect to each $\boldsymbol{W}_{N_h}$, however as mentioned this is inefficient as each set of parameters will have a unique update equation, also many of the calculations for each of these differentiations are repeated. On top of that this process is not universal for all neural networks. Backpropagation is a way to compute the gradients in a systematic fashion to efficiently calculate all the gradients in a neural network one layer at a time that can be universally applied to all neural networks that also minimizes the amount of duplicate calculations. At a high level backpropagation can be thought of as a large chain rule. The per-layer loss gradient, often called the local gradient, is computed backwards across layers of the network. In this manner the local gradient for layer i is computed with respect to only the inputs and outputs of layer i. When applied in a chain-rule like manner the loss is passed backwards, starting at the output, through each layer. Each layers' parameters are updated in accordance with how much those parameters attribute, via the gradient, to the total loss. For detailed explanation please see [31].

CURRENT STATE OF QUANTUM COMPUTING

Quantum Circuit Architecture

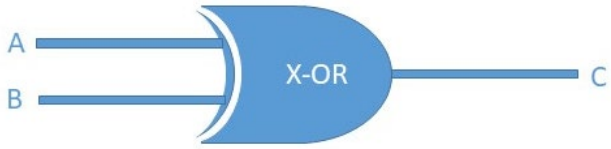
The currently dominant approach to quantum computing is to create a quantum circuit. This approach is similar to classical digital logic circuits in organization and structure, but there are some key differences. In classical digital logic circuits, a set of bits are typically initialized in the binary 0 state, and are fed through logic gate operations sequentially until the computation is complete and the results are read out. These circuits are often represented graphically by sequences of lines between symbols representing logic gates that eventually lead to an output. Critically, the values of the bits of the circuit can be measured at any point throughout the circuit without affecting the rest of the circuit.

With quantum circuits, qubits are similarly prepared in some initial state, usually the qubit's zero state, and are also fed through sequences of gate operations that are also graphically represented by symbols (typically rectangles) connected by lines that eventually lead to some output which is read by quantum measurement. However a major difference from classical digital circuits is that a qubit measured before the end of the circuit will have significant effects on the rest of the circuit. This is because there is an associated back-action as a result of any quantum measurement, and often the measurement back-action collapses the quantum wave function of the measured qubit, reducing it to a single classical value from that point on. This aspect is represented graphically in quantum circuits using double lines for classical values and single lines for quantum values.

Another major difference from digital logic is regarding circuit structure. As described in the fundamentals of quantum computing section, digital logic gates are allowed to have a different number of inputs than outputs, while quantum gates must be unitary and thus have equal numbers of inputs and outputs. For example classical gates such as AND, OR, and XOR gates have two inputs and only one output, such that the operations are irreversible and the total number of bits at any given point in the circuit is not fixed. Since quantum gate operations must be unitary and reversible, the total number of qubits is conserved throughout the circuit[3]. For example, consider the classical XOR digital logic gate and the quantum CNOT gate. These two gates have similar outputs that produce similar truth tables. For the digital XOR gate shown below, two inputs, A and B, are fed in and one output, C, is produced. If A and B are the same, the output is a value of 0. If A and B are different, the output is a value of 1.

Input		Output
A	B	C

[3] Note that quantum measurement performed before the end of the circuit may often be graphically represented as reducing the number of qubits, however these qubits continue to exist classically after measurement.



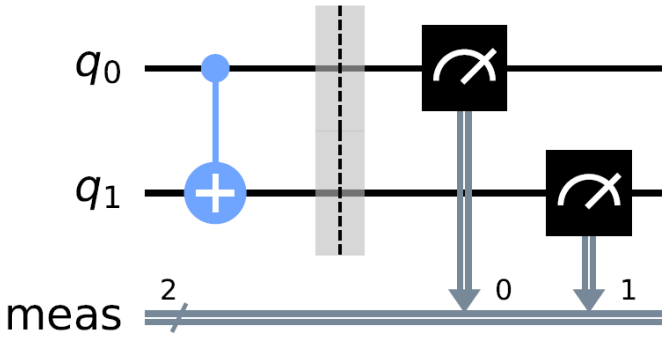
0	0	0
0	1	1
1	0	1
1	1	0

Figure 9: The XOR gate

In contrast, the quantum CNOT gate uses the state of one of the input qubits as a control qubit, and determines the action on the other qubit, the target qubit, based on the control qubit's state. This is represented graphically in Figure 10, where q_0 is the control qubit and q_1 is the target qubit. The CNOT gate itself is represented by a unique symbol. The $\oplus$ symbol represents the NOT operation being applied to q_1, which is connected to the q_0 qubit and terminates in a dot representing q_0 as the control of the NOT operation. If q_0 is in the zero state, q_1 is unaffected, while if q_0 is in the one state, the NOT operation will be applied to q_1 and its state will be flipped, which is equivalent to rotation by π about the x-axis. In contrast to the digital XOR gate, both input qubits are conserved throughout the calculation and are measured at the end of the circuit.

Also unique to quantum computing is that the output of the quantum measurement process is not the quantum state. Instead, an observable associated with the qubit is measured, yielding one of the possible eigenvalues of the operator corresponding to the the quantum state the qubit was in. This is highlighted in the truth table in Figure 10, where the inputs are quantum states represented in ket notation, and the outputs are the measured eigenvalues of the operator, and associated with the two possible quantum state outcomes.

Input		**Output**	
q_0	q_1	M_0	M_1
|0>	|0>	0	0
|0>	|1>	0	1



|1>	|0>	1	1
|1>	|1>	1	0

Figure 10: The quantum CNOT gate

In this diagram, M_0 is the measurement outcome on q_0, and M_1 is the measurement outcome on q_1. If we omit the M_0 column, then we recover the truth table for the classical XOR gate. Note however that this truth table does not include the continuum of possible superpositions of qubit states, which are valid inputs in the analogous quantum gate. Additionally, the q_1 qubit is still present at the end of the circuit which, for unitary gates, allows for reversibility and reconstruction of the input states given the output and operation applied. Graphically, gates are applied sequentially from left to right, as depicted below.

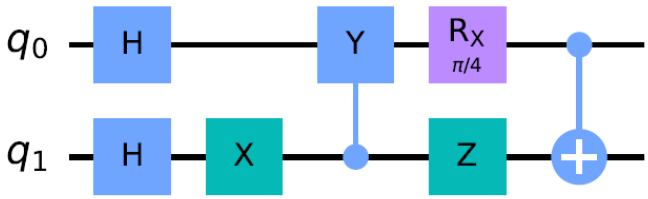


Figure 11: Simple quantum circuit example

There are some gates that operate on larger numbers of qubits. They can be generically represented graphically by rectangles that cover multiple qubit lines, however some specific multi-qubit gates have their own representations. Any unitary single qubit operation can be turned into a controlled operation that depends on the state of another qubit. This is shown by the appropriate box/symbol on the qubit to be operated on, with a vertical line extending from the box to the horizontal line of the controlling qubit with a dot placed at their intersection.

Another contrast to classical logic circuits is that in quantum circuits, one measurement is not enough to deduce the quantum state of the output qubit(s). For example, a qubit in the superposition state

$$|q\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad (79)$$

has a measurement outcome of 0 with probability ½ and a measurement outcome of 1 with probability ½. Thus several measurements of the same qubit must be made in order to deduce

that it is in the given superposition state. Thus the expectation value is estimated by repeating the measurement many times. Also, since the measurement value only returns the magnitude, the expectation value will be equivalent for a set of quantum states that have the same magnitude but different phase, for example the state

$$|q\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}. \quad (80)$$

These two states are identical except for a phase factor, and this type of measurement protocol (with a single Z measurement) does not have the resolution to discern between such states. There are workarounds, for example measuring with respect to the x-axis instead of the z-axis, but this leads to extra care that is necessary in designing quantum circuits. For further reading on the fundamentals quantum gates and quantum information, see references [18] and [19].

Embedding Classical Data in Quantum Circuits

Since quantum data and classical data are inherently different in nature, methods must be used to encode classical data in a way that is usable in quantum circuits. Currently, there are two main strategies for building quantum machine learning circuits that use classical data. The first strategy is to use classical dimensionality reduction techniques to reduce the dimension of the classical data to match the number of qubits available in the circuit, such as principle component analysis. In order to embed binary data specifically, an additional step is necessary to convert the reduced dimension data to binary values. An example of a method to reduce dimensionality and convert to binary values is shown in Appendix A.

Another type of encoding is often called gate encoding. In this paradigm the original floating point data is encoded directly into a quantum circuit with the use of rotation gates. For this type of encoding, the original data is normalized to be in the range of $[0, \pi]$. This range is used to ensure that large and small values are not unintentionally confused for being close together, as they could be if the full $[0, 2\pi)$ range was used. In cases where the data has fewer or the same dimensions as the number of qubits, each value of the original data can be directly encoded into the rotation parameter of a rotation gate. In this setup each dimension of the data is encoded by exactly one rotation gate per qubit during the encoding, which can then be used by additional circuit elements for machine learning.

An extension of this method, called block encoding, takes this method and applies it to higher dimensional data. Here the data is represented as a quantum circuit containing many rotation gates applied to the same qubits in order to generate a unique encoding for each data point. Four numbers are important for the design of this encoding scheme: the dimensionality of the data (D), the number of qubits (Q), the number of layers of the circuit (L), and the number of gates per block (G). The values of D and Q should already be known and the values of L and G are design variables. To select the values of L and G follow the rule that $D \leq QLG$ while also trying to minimize the product QLG. For example, if the data consists of 192 total dimensions, and we

are using a quantum computer/simulator with 16 qubits, we can set L to 4 and G to 3. With the design settled, the circuit can be created.

The circuit will consist of creating blocks of sets of cycling rotation gates (i.e. an x rotation gate followed by a z rotation gate followed by another x rotation gate). Consecutive rotation gates must be around different axes. The circuit is created by stacking these blocks together evenly across all qubits. After a layer of blocks is created a series of CNOT gates are used to connect consecutive qubit pairs. This process is repeated L times. The rotation amount is defined by the data itself as was done with the gate encoding above. If there are more gates than data dimensions the excess rotation gates use 0 for the rotation angle, so they act as pass through gates. The outputs from this encoding circuit now encode the full data and return unique values for each of the inputs, without needing to follow a complicated dimensionality reduction technique. An example of a circuit with eight qubits, four layers, and two gates per block is shown below in Figure 12.

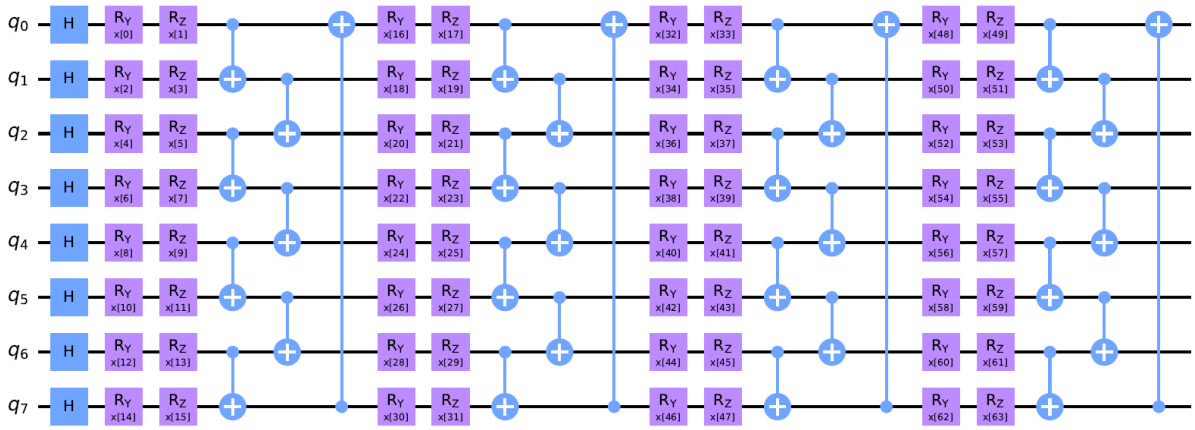


Figure 12: Example of block encoding method

Quantum Machine Learning

Quantum machine learning is a new and emerging sub-field of quantum computing that combines two specialized fields into one. The overall process of quantum machine learning is actually very similar to machine learning on classical computers, since quantum machine learning is really a hybrid quantum-classical computation [32]. In quantum machine learning a few key components of the classical machine learning process are replaced by the output of a quantum computer. Most importantly, the error function to be optimized is at least partially calculated by a quantum computer. At least one expectation value of a qubit of a quantum circuit is used to compose the error function [33], though classical components may be included as well, which in some cases increases the functionality. For example, in a classification problem the correct label will be a purely classical value, while the label predicted by the network is calculated on a quantum computer. Also in order to be able to tune and train the quantum network, the network must include some classical parameters that can be kept track of and updated by the algorithm [32] [33].

Once the data has been encoded into the quantum circuit using one of the encoding methods mentioned, multi-qubit operations are applied to the data qubits and the readout qubits with the goal of manipulating the readout qubits to some desired state corresponding to the data input. Typically these operations are parametrized controlled rotation gates applied to the readout qubit

and controlled off of the data qubits, though non-controlled gates can also be applied to the readout qubit. Upon running the circuit multiple times, the expectation value of the measured readout qubit is used as the final output of the circuit and used to compute a loss function. An example circuit for quantum machine learning is shown below, this example used the single rotation gate data encoding method mentioned above.

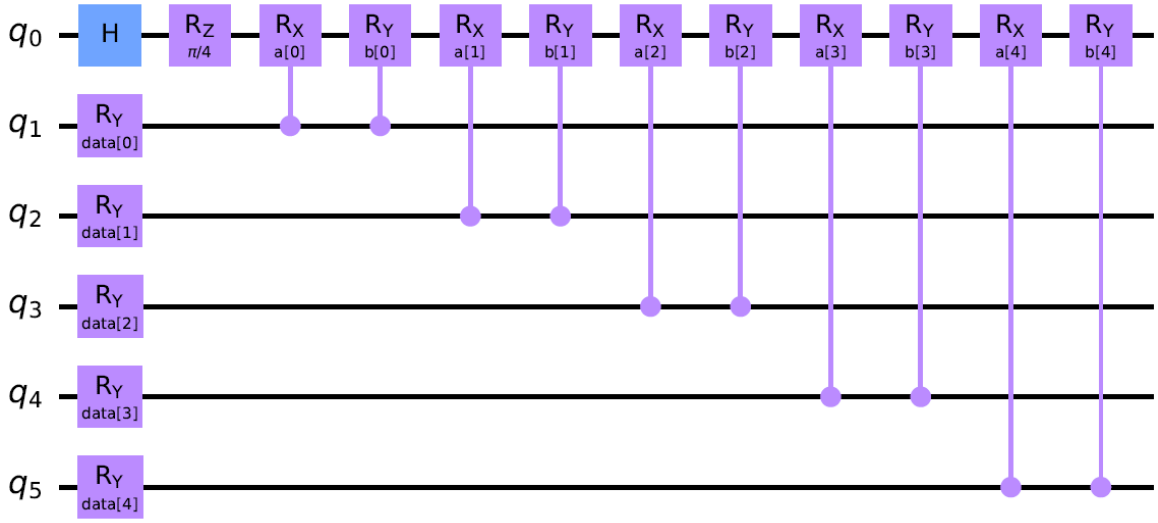


Figure 13: Example quantum machine learning circuit

In this setup, qubit 0 is the readout qubit and the only one that is measured for the output. The first two gates acting on qubit 0 place the qubit into an unbiased initial state before the network operations are applied. Qubits 1-5 are the data qubits. Each rotation operation on each of those qubits is parameterized by some classical value based on the input data. The remaining gates on qubit 0 are trainable rotation gates controlled by the state of the data qubits. The network gates are parameterized by the trainable network variables $a[1], b[1], a[2], b[2], \ldots, a[n], b[n]$.

Similar to classical machine learning, the most common method used to find the optimal parameter values in quantum machine learning is a variant of gradient descent [32]. Since at least part of the error function is calculated on a quantum computer, the gradient calculation also requires partial computation on a quantum computer, which leads to another major difference between classical and quantum machine learning. In classical machine learning fast gradient calculation is enabled by backpropagation. Backpropagation requires intermediate results to be measured/calculated and stored for later use to avoid recalculating them many times. Obtaining intermediate results in the calculation on a quantum computer would require intermediate measurements. However, on a quantum computer these intermediate measurements would destroy any quantum behavior being utilized by the quantum computer. This means that in order to maintain any true quantum calculation, backpropagation is not possible [32] [34] and other methods must be used for calculating the gradient on quantum computers [34]. Fortunately, other methods of calculating the gradient called parameter-shift rules have been developed and fit very well into the architecture of quantum computing.

Parameter-Shift Rule

The parameter-shift rule is a very useful tool that allows for an analytically exact gradient calculation that can be performed on a quantum computer using the same circuit used to

calculate the loss, but with shifted parameter values [34] [35]. In practice, this results in an approximate gradient due to the approximation of the expectation value. This is also the case in the classical machine learning context defined above, however in that case we defined the loss as the finite approximation (under finite data) to the true expectation value. However, in the quantum machine learning context, there are in effect two expectations: an expectation with respect to measurement outcomes and an expectation over the data. In contrast to the finite data problem, expectation values over measurement outcomes can be run as many times as necessary to give sufficient precision.

To show the derivation of this rule, a generic loss function in the form of an expectation value from a readout qubit will be used. Let the loss function $C(\theta)$ be defined as an expectation value [34] [35] [36]:

$$C(\theta) := \langle\psi|\hat{U}_G^\dagger(\theta)\,\hat{A}\,\hat{U}_G(\theta)|\psi\rangle \tag{81}$$

where $|\psi\rangle$ is the vector representing the quantum state and $\langle\psi|$ is its complex conjugate transpose, $\hat{U}_G(\theta)$ is a unitary operator parameterized by θ with the form $\hat{U}_G(\theta) = e^{-ia\theta\hat{G}}$ where $\hat{G}$ is a Pauli operator, $\hat{U}_G^\dagger(\theta)$ is the complex conjugate transpose of $\hat{U}_G(\theta)$, $\hat{A}$ is the observable being measured, and a is a fixed constant. Taking the derivative with respect to the parameter θ:

$$\frac{dC(\theta)}{d\theta} = \frac{d}{d\theta}\langle\psi|\hat{U}_G^\dagger(\theta)\,\hat{A}\,\hat{U}_G(\theta)|\psi\rangle, \tag{82}$$

requires the use of the product rule, giving:

$$\begin{aligned}
\frac{dC(\theta)}{d\theta} &= \left\langle\psi\right|\frac{d}{d\theta}(\hat{U}_G^\dagger(\theta))\,\hat{A}\,\hat{U}_G(\theta)\left|\psi\right\rangle + \left\langle\psi\right|\hat{U}_G^\dagger(\theta)\,\hat{A}\frac{d}{d\theta}(\hat{U}_G(\theta))\left|\psi\right\rangle \\
\frac{dC(\theta)}{d\theta} &= \langle\psi|\,(ia\hat{G})\hat{U}_G^\dagger(\theta)\,\hat{A}\,\hat{U}_G(\theta)|\psi\rangle + \langle\psi|\,\hat{U}_G^\dagger(\theta)\,\hat{A}\,\hat{U}_G(\theta)(-ia\hat{G})|\psi\rangle \\
\frac{dC(\theta)}{d\theta} &= ia\big(\langle\psi|\,\hat{U}_G^\dagger(\theta)\,\hat{G}\hat{A}\,\hat{U}_G(\theta)|\psi\rangle - \langle\psi|\,(\hat{U}_G^\dagger(\theta)\,\hat{A}\,\hat{G}\hat{U}_G(\theta)|\psi\rangle\big) \\
\frac{dC(\theta)}{d\theta} &= ia\langle\psi|\,\hat{U}_G^\dagger(\theta)\,[\hat{G},\hat{A}]\hat{U}_G(\theta)|\psi\rangle,
\end{aligned} \tag{83}$$

where $[\hat{G},\hat{A}]$ is the commutator $\hat{G}\hat{A} - \hat{A}\hat{G}$. While having a commutator in the calculation seems to complicate things, it does allow for the following identity to be used [35] [36] [37]:

$$[\hat{G},\hat{A}] = -i\left(\hat{U}_G^\dagger\left(\frac{\pi}{2}\right)\,\hat{A}\,\hat{U}_G\left(\frac{\pi}{2}\right) - \hat{U}_G^\dagger\left(-\frac{\pi}{2}\right)\,\hat{A}\,\hat{U}_G\left(-\frac{\pi}{2}\right)\right), \tag{84}$$

where $\hat{U}_G = e^{-ia\theta\hat{G}}$, and $\hat{G}$ is assumed to be some Pauli operator. Using this identity does limit the application of the final result to be valid only with Pauli operator-based unitary gates; but

with how commonly used Pauli gates are, this result is still applicable. For a proof of this identity, see Appendix B.

Applying this identity to the commutator in $ia\langle\psi|\,\hat{U}_G^\dagger(\theta)\,[\hat{G},\hat{A}]\hat{U}_G(\theta)|\psi\rangle$ leads to a form more compatible with quantum circuits:

$$\frac{dC(\theta)}{d\theta} = ia\left\langle\psi\right|\hat{U}_G^\dagger(\theta)(-i)\left(\hat{U}_G^\dagger\left(\frac{\pi}{2}\right)\hat{A}\,\hat{U}_G\left(\frac{\pi}{2}\right) - \hat{U}_G^\dagger\left(-\frac{\pi}{2}\right)\hat{A}\,\hat{U}_G\left(-\frac{\pi}{2}\right)\right)\hat{U}_G(\theta)\left|\psi\right\rangle$$

$$\frac{dC(\theta)}{d\theta} = a\left(\left\langle\psi\right|\hat{U}_G^\dagger(\theta)\hat{U}_G^\dagger\left(\frac{\pi}{2}\right)\hat{A}\hat{U}_G(\theta)\,\hat{U}_G\left(\frac{\pi}{2}\right)\left|\psi\right\rangle - \left\langle\psi\right|\hat{U}_G^\dagger(\theta)\hat{U}_G^\dagger\left(-\frac{\pi}{2}\right)\hat{A}\,\hat{U}_G(\theta)\,\hat{U}_G\left(-\frac{\pi}{2}\right)\left|\psi\right\rangle\right)$$

$$\frac{dC(\theta)}{d\theta} = a\left(\left\langle\psi\right|\hat{U}_G^\dagger\left(\theta+\frac{\pi}{2}\right)\hat{A}\,\hat{U}_G\left(\theta+\frac{\pi}{2}\right)\left|\psi\right\rangle - \left\langle\psi\right|\hat{U}_G^\dagger\left(\theta-\frac{\pi}{2}\right)\hat{A}\,\hat{U}_G\left(\theta-\frac{\pi}{2}\right)\left|\psi\right\rangle\right) \quad (85)$$

In this form, each term is an expectation value so it can be calculated by a quantum circuit. Of even more importance to this application is that each term is in the same form as the original loss function $C(\theta)$ except for the shift by $\pm\pi/2$. This means the gradient calculation can utilize the exact same circuit as the original loss function. For each parameter's gradient calculation all that is required is running the circuit twice more, once with the parameter shifted up by $\pi/2$, and once with the parameter shifted down by $\pi/2$ [35]. Using the circuit given in Figure 11 as an example, to calculate the gradient for the first gate parameter, $a[0]$, and letting $k = \pi/2$, the circuits in Figure 14 would both be run and the output measured for each circuit.

The difference between the outputs of the two circuits in Figure 14 and the factor of a can be calculated classically in the hybrid quantum-classical scheme, which will then give the gradient necessary for gradient descent optimization without requiring intermediate measurements nor interrupting the quantum behavior of the quantum computer. The gradient calculation process is repeated for every network parameter, and the parameters are updated according to the gradient result. With a way to efficiently calculate gradients on a quantum computer, the overall quantum machine learning process can be described by Algorithm 2.

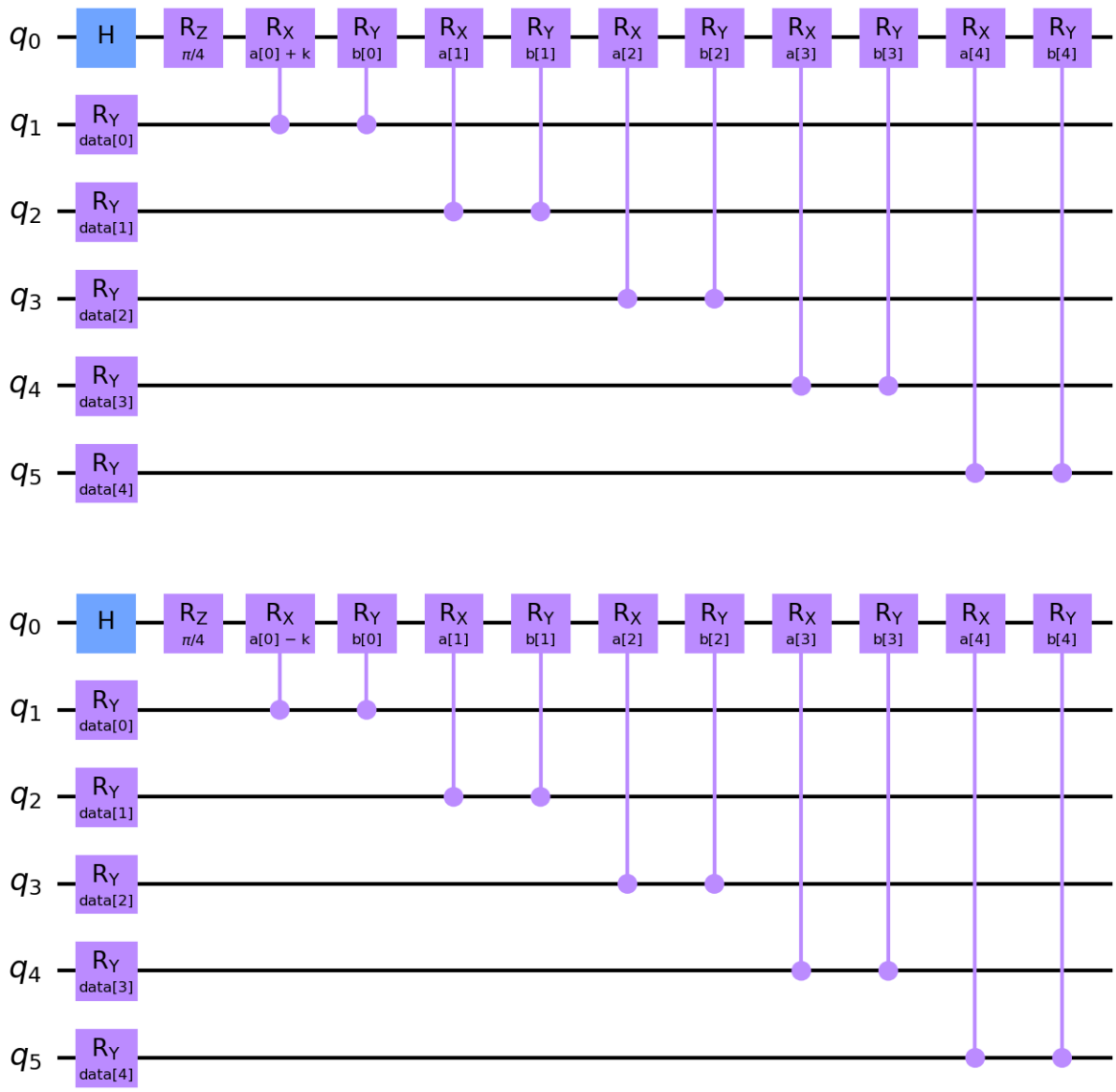


Figure 14: Circuits for parameter a1 upshift (top) and downshift (bottom)

Algorithm 2: Process for quantum machine learning

Inputs: Training data $\boldsymbol{X_i, \ldots, X_m}$, training labels $\boldsymbol{y}$, randomly initialized parameters $w_1, \ldots, w_n$, learning rate *r,* shift *k*

1. For $\boldsymbol{i = 1, \ldots, m}$
   1.1. Run circuit with parameters $\mathbf{w} = [\boldsymbol{w_1, \ldots, w_n}]$, and calculate the expectation value of the output $\langle \widehat{\boldsymbol{A}}(\boldsymbol{X_i, w}) \rangle$
   1.2. Calculate loss using data label and circuit output $\boldsymbol{C(y_i, X_i, w) = y_i - \langle \widehat{A}(X_i, w) \rangle}$
   1.3. For $\mathbf{j = 1, \ldots, n}$ do
      1.3.1. Upshift $\boldsymbol{j}^{\text{th}}$ parameter: $\boldsymbol{w_+ = [w_1, \ldots, w_j + k, \ldots, w_n]}$
      1.3.2. Run circuit with new parameter set $\boldsymbol{w_+}$ and measure output $\boldsymbol{upshift = \langle \widehat{A}(X_i, w_+) \rangle}$
      1.3.3. Downshift $\boldsymbol{j}^{\text{th}}$ parameter: $\boldsymbol{w_- = [w_1, \ldots, w_j - k, \ldots, w_n]}$
      1.3.4. Run circuit with new parameter set $\boldsymbol{w_-}$ and measure output $\boldsymbol{downshift = \langle \widehat{A}(X_i, w_-) \rangle}$
      1.3.5. Calculate gradient with respect to $\boldsymbol{j}^{\text{th}}$ parameter $\boldsymbol{\nabla_{w_j} C(y_i, X_i, w) = \frac{1}{a}(upshift - downshift)}$
   1.4. Update parameter $\boldsymbol{w_j = w_j - r\nabla_{w_j} C(y_i, X_i, W)}$

Stochastic Parameter-Shift Rule

The stochastic parameter-shift rule allows for a more generalizable gradient calculation that is applicable to a wider variety of gates including multi-qubit gates [38]. This is done by replacing the operator $\hat{G}$ in $\widehat{U}_G(\theta) = e^{-ia\theta\hat{G}}$ with $\hat{G}(\theta) = \widehat{H} + \theta\widehat{V}$, where $\widehat{H}$ is an arbitrary linear combination of Pauli operator tensor products, and $\widehat{V}$ is a tensor product of Pauli operators. Since multi-qubit operators can be constructed as a sum of tensor products of Pauli operators, the use of $\widehat{H}$ and $\widehat{V}$ in this form allows for generalization to arbitrary gates and calculation of the gradient analytically. The loss function $C(\theta)$ then becomes:

$$C(\theta) = \langle\psi|\widehat{U}_G^{\dagger}(\theta)\, \hat{A}\, \widehat{U}_G(\theta)|\psi\rangle = \langle\psi|e^{ia(\widehat{H}+\theta\widehat{V})}\, \hat{A}\, e^{-ia(\widehat{H}+\theta\widehat{V})}\, |\psi\rangle. \quad (86)$$

To find the gradient of this loss function and manipulate it into a form compatible with quantum computers requires several identities. The Baker-Campbell-Hausdorff (BCH) identity [39] is derived in Appendix C, and is given by:

$$f(\lambda) = e^{\lambda\hat{A}}\hat{B}e^{-\lambda\hat{A}} = \left(\sum_{n=0}^{\infty} \frac{(\lambda)^n[\hat{A},\cdot]^n}{n!}\right)\hat{B} = e^{\lambda[\hat{A},\cdot]}\hat{B} \quad (87)$$

We also apply the commutator identity in Equation (84), and the following exponential derivative rule (derived in Appendix D) [38] [39]:

$$\frac{\partial \mathrm{e}^{\hat{z}(\theta)}}{\partial \theta} = \int_0^1 e^{(1-s)\hat{z}(\theta)} \frac{\partial \hat{z}(\theta)}{\partial \theta} \mathrm{e}^{s\hat{z}(\theta)} \, \mathrm{d}s$$

(88)

With this, we derive an analytic form of the gradient that may still be evaluated by a quantum computer. The starting point is the loss function as before, given by:

$$C(\theta) = \langle\psi|\hat{U}_G^\dagger(\theta)\, \hat{A}\, \hat{U}_G(\theta)|\psi\rangle = \langle\psi|e^{ia(\hat{H}+\theta\hat{V})}\, \hat{A}\, e^{-ia(\hat{H}+\theta\hat{V})}\, |\psi\rangle$$

(89)

Applying the BCH identity (87) yields:

$$C(\theta) = \langle\psi|e^{ia[\hat{H}+\theta\hat{V},\cdot]}\, \hat{A}\, |\psi\rangle$$

(90)

Next, the derivative is taken and passed into the expectation:

$$\frac{dC(\theta)}{d\theta} = \frac{\partial}{\partial\theta}\langle\psi|e^{ia[\hat{H}+\theta\hat{V},\cdot]}\, \hat{A}\, |\psi\rangle = \left\langle\psi\right| \frac{\partial}{\partial\theta} e^{ia[\hat{H}+\theta\hat{V},\cdot]}\, \hat{A}\, \left|\psi\right\rangle$$

(91)

From here the exponential derivative rule in Equation (88) is applied, where $e^{\hat{z}(\theta)} = e^{ia[\hat{H}+\theta\hat{V},\cdot]}$ and $\frac{\partial\hat{z}(\theta)}{\partial\theta} = \frac{\partial}{\partial\theta} ia\left[\hat{H}+\theta\hat{V},\cdot\right] = ia[\hat{V},\cdot]$, giving:

$$\frac{dC(\theta)}{d\theta} = \left\langle\psi\right| \frac{\partial}{\partial\theta} e^{ia[\hat{H}+\theta\hat{V},\cdot]}\, \hat{A}\, \left|\psi\right\rangle = \left\langle\psi\right| \int_0^1 e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]} ia[\hat{V},\cdot] \mathrm{e}^{sia[\hat{H}+\theta\hat{V},\cdot]} \hat{A}\, \mathrm{d}s \left|\psi\right\rangle$$

(92)

Next, the constants are moved to the front and BCH is applied to the term inside the red curly braces:

$$\begin{aligned}\frac{dC(\theta)}{d\theta} &= ia\left\langle\psi\right| \int_0^1 e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]} [\hat{V},\cdot]\{\mathrm{e}^{sia[\hat{H}+\theta\hat{V},\cdot]}\hat{A}\}\, \mathrm{d}s \left|\psi\right\rangle \\ &= ia\left\langle\psi\right| \int_0^1 e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]} [\hat{V},\cdot] \mathrm{e}^{sia(\hat{H}+\theta\hat{V})}\hat{A}\mathrm{e}^{-sia(\hat{H}+\theta\hat{V})}\, \mathrm{d}s \left|\psi\right\rangle \\ &= ia\left\langle\psi\right| \int_0^1 e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]} \left[\hat{V}, \mathrm{e}^{sia(\hat{H}+\theta\hat{V})}\hat{A}\mathrm{e}^{-sia(\hat{H}+\theta\hat{V})}\right] \mathrm{d}s \left|\psi\right\rangle,\end{aligned}$$

(93)

where we use the commutator notation $\left[\hat{A},\cdot\right]\hat{B} = \left[\hat{A},\hat{B}\right]$. Since the commutator now only contains $\hat{V}$, we can now apply the commutator identity (87):

$$\frac{dC(\theta)}{d\theta} = ia\left\langle\psi\right| \int_0^1 e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]} (-i) \left(\hat{U}_V^\dagger\left(\frac{\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})}\hat{A}\mathrm{e}^{-sia(\hat{H}+\theta\hat{V})}\hat{U}_V\left(\frac{\pi}{2}\right)\right.$$

$$\left. -\hat{U}_V^\dagger\left(\frac{-\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})}\hat{A}\mathrm{e}^{-sia(\hat{H}+\theta\hat{V})}\hat{U}_V\left(\frac{-\pi}{2}\right)\right) \mathrm{d}s \left|\psi\right\rangle$$

$$= a\left\langle \psi \left| \int_0^1 e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]} \left(\hat{U}_V^\dagger\left(\frac{\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{\pi}{2}\right) \right.\right.\right.$$

$$\left.\left.\left. -\hat{U}_V^\dagger\left(\frac{-\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{-\pi}{2}\right) \right) ds \right| \psi \right\rangle,$$

(94)

where $\hat{U}_V(\theta) = e^{-ib\theta\hat{V}}$ and b is another constant $b \neq a$. The BCH identity is applied once more to the entire term inside the brackets where

$$\left(\hat{U}_V^\dagger\left(\frac{\pi}{2}\right) e^{sia(\hat{H}+\theta\hat{V})} \hat{A} e^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{\pi}{2}\right) - \hat{U}_V^\dagger\left(\frac{-\pi}{2}\right) e^{sia(\hat{H}+\theta\hat{V})} \hat{A} e^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{-\pi}{2}\right) \right)$$

(95)

is treated as the matrix $\hat{B}$ in $e^{\lambda\hat{A}}\hat{B}e^{-\lambda\hat{A}} = e^{\lambda[\hat{A},\cdot]}\hat{B}$, and $e^{(1-s)ia[\hat{H}+\theta\hat{V},\cdot]}$ is the exponential. This BCH identity application yields:

$$\frac{dC(\theta)}{d\theta} = a\left\langle \psi \left| \int_0^1 e^{(1-s)ia(\hat{H}+\theta\hat{V})} \left(\hat{\mathrm{U}}_V^\dagger\left(\frac{\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{\mathrm{U}}_V\left(\frac{\pi}{2}\right) \right.\right.\right.$$

$$\left.\left.\left. -\hat{U}_V^\dagger\left(\frac{-\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{-\pi}{2}\right) \right) e^{-(1-s)ia(\hat{H}+\theta\hat{V})} ds \right| \psi \right\rangle$$

$$= a\left(\left\langle \psi \right| \int_0^1 e^{(1-s)ia(\hat{H}+\theta\hat{V})} \hat{U}_V{}^\dagger\left(\frac{\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{\pi}{2}\right) e^{-(1-s)ia(\hat{H}+\theta\hat{V})} \, \mathrm{d}s \left| \psi \right\rangle \right.$$

$$\left. - \left\langle \psi \right| \int_0^1 e^{(1-s)ia(\hat{H}+\theta\hat{V})} \hat{U}_V^\dagger\left(\frac{-\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{-\pi}{2}\right) e^{-(1-s)ia(\hat{H}+\theta\hat{V})} \, \mathrm{d}s \left| \psi \right\rangle \right)$$

$$= a\int_0^1 \left(\left\langle \psi \left| e^{(1-s)ia(\hat{H}+\theta\hat{V})} \hat{U}_V^\dagger\left(\frac{\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{\pi}{2}\right) e^{-(1-s)ia(\hat{H}+\theta\hat{V})} \right| \psi \right\rangle \right.$$

$$\left. - \left\langle \psi \left| e^{(1-s)ia(\hat{H}+\theta\hat{V})} \hat{U}_V^\dagger\left(\frac{-\pi}{2}\right) \mathrm{e}^{sia(\hat{H}+\theta\hat{V})} \hat{A} \mathrm{e}^{-sia(\hat{H}+\theta\hat{V})} \hat{U}_V\left(\frac{-\pi}{2}\right) e^{-(1-s)ia(\hat{H}+\theta\hat{V})} \right| \psi \right\rangle \right) ds,$$

(96)

which is the stochastic parameter-shift rule [38] [39]. Using the BCH identity, the gradient was able to be manipulated back into the form of a difference of two expectation values with the observable $\hat{A}$ at the center, the gate operations and state vector $|\psi\rangle$ on its right, and their complex conjugates transposed on its left. Comparing (96) to the original loss function, $C(\theta) = \langle\psi| e^{ia(\hat{H}+\theta\hat{V})}\ \hat{A}\ e^{-ia(\hat{H}+\theta\hat{V})}\ |\psi\rangle$, the gradient contains more terms as well as an integral. In order to perform this gradient calculation, a second quantum circuit that represents the gradient calculation would need to be set up and run in conjunction with the original circuit used to calculate the loss [38]. Additionally, the integral is approximated by the following sampling scheme:

$$\int_0^1 \langle\psi|\, F(s)|\psi\rangle \approx \frac{1}{M}\sum_i^M \langle\psi|F(s_i)|\psi\rangle\,,\ \text{with } s_i \sim U(0,1)$$

(97)

A value for s is randomly sampled from a uniform distribution for each run of the circuit [38] and expectations are averaged. While this does require more resources to run, it is an accurate quantum calculation of the gradient of the quantum loss function.

Quantum Computing Example – The XOR Problem

The exclusive-or (XOR) problem was discussed earlier in the introduction to artificial neural networks. The problem consists of two classes of data on a grid separated into four blocks, where blocks diagonal from each other contain points in the same class, as depicted in Figure 5. This results in a classification problem where the two classes are not linearly separable. Comparing and contrasting the classical and quantum solutions highlights some of the advantages of quantum computing. Due to the non-linear separation between classes, a classical neural network requires multiple perceptrons to solve the XOR problem. However, it has been shown that a simple quantum circuit, shown in Figure 15, using only one qubit as a single perceptron can solve the XOR problem. This approach leverages the phase of the qubit as an extra degree of freedom [40].

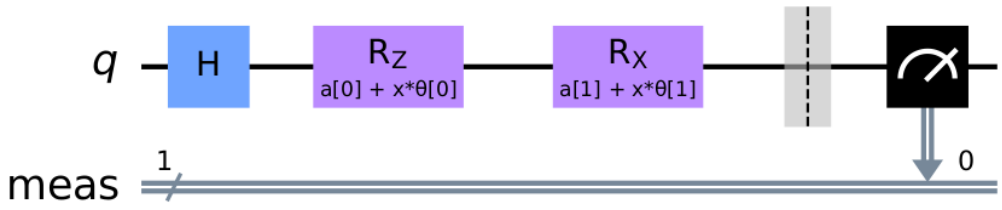


Figure 15: Quantum circuit to solve XOR problem

This circuit is fairly simple and consists of only three gates, a Hadamard gate followed by a Z-rotation gate, and then an X-rotation gate. The rotation angles of the gates are determined by the following classical expressions [40]:

$$\text{Z-rotation angle: } \theta_1 x_1 + \alpha$$

$$\text{X-rotation angle: } \theta_2 x_2 + \alpha$$

(98)

In these expressions, θ_1 and θ_2 are trainable parameters, x_1 and x_2 are the input values, and α is another trainable parameter. Here, there is a direct solution by using $\theta_1 = \theta_2 = \pi$ and $\alpha = -\pi/2$ [40], however this circuit could be trained by gradient descent. Indeed, for the input vectors (0, 0) and (1, 1) the circuit gives a result near the zero state (up to quantum hard precision), and for the input vectors, (0, 1) and (1, 0) the circuit gives an output near the one state (up to quantum hard precision) [40]. Finally, with "noisy" non-integer inputs between 0 and 1 the circuit gives output states between the zero state and one state (see Table 2 in [40]).

Extending the circuit and methods introduced above, nearly identical results are obtained using a quantum machine learning framework and training on a larger data set. To do this, the circuit from [40] is modified to make it more compatible with the parameter-shift gradient descent method described earlier. This modified circuit is shown below in Figure 16.

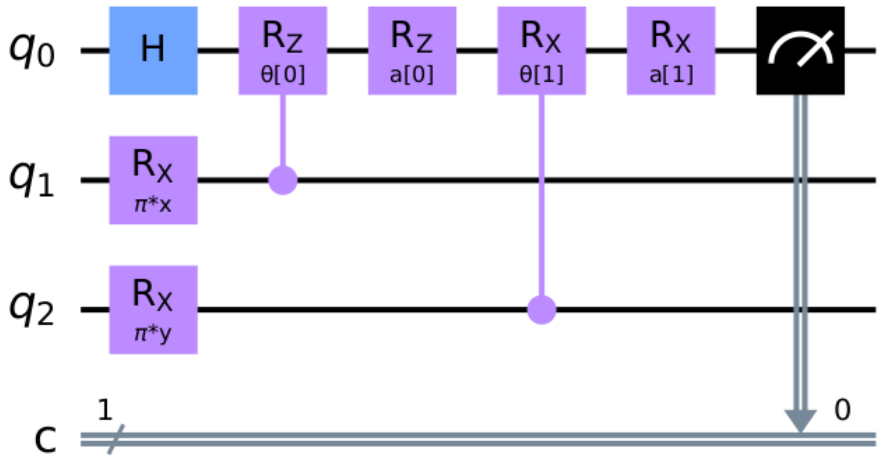


Figure 16: Modified XOR circuit used in training

This circuit added two qubits so that the rotation data embedding scheme can be used. The initial Hadamard gate applied to the readout qubit remains that same. The CRZ(θ_1) controlled on Qubit 1 for the first data input is an equivalent representation of the $\theta_1 x_1$ part of the input parameters to the original Z rotation gate, and similarly for CRX(θ_2) and $\theta_2 x_2$. Since originally the α parameter is added as a constant, it can be applied in the new circuit as another gate applied in series with the respective controlled gate. Additionally, the single α parameter has been split into α_1 and α_2 for the Z and X rotations, respectively, to allow the circuit to be more flexible.

The data for training and testing this new circuit is generated from a random uniform distribution between 0 and 1 for the x and y values of each data point, though values of exactly 0.5 were excluded as they would be on the class boundary and degenerate. 1000 sample points were generated, with 750 being used for training, 63 used for validation during training, and 187 used for blind testing after training was complete. This dataset is shown in Figure 17.

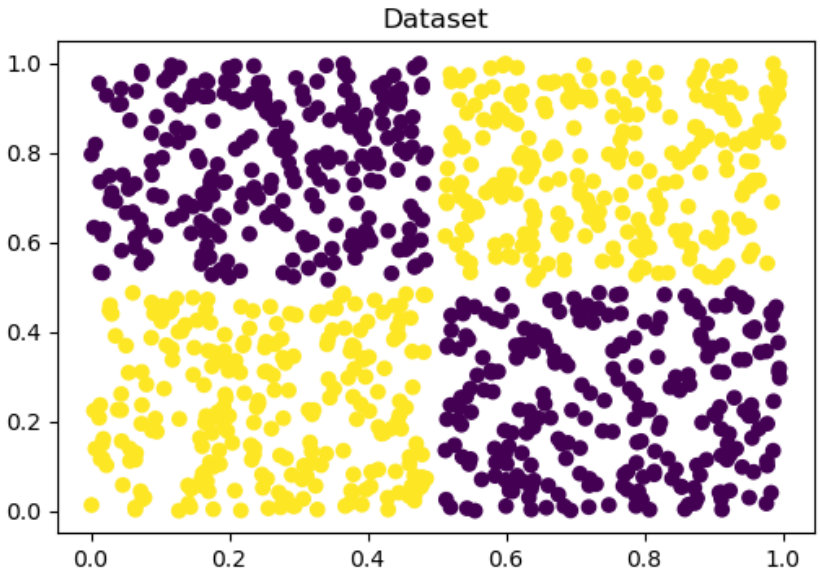


Figure 17: Generated XOR dataset

The yellow data points are assigned to class 0, corresponding to a zero state output of the read-out qubit, and the purple data points are assigned to class 1, corresponding to a one state output of the read-out qubit. The observable used is the Pauli Z gate, which has two possible eigenvalues {+1,-1}, which are used as the labels for the classes, respectively. The goal for the network is to rotate the read-out qubit towards the zero state for data from class 0, and towards the one state for data from class 1. The expectation value should be closer to +1 for inputs from class 0, and closer to -1 for inputs from class 1. The loss function to be optimized is given by the mean squared error of the expectation value.

To optimize the loss function, mini-batch gradient descent optimization was used, with a batch size of 25 data samples and a learning rate (or step size) of 0.025. The network was trained over 150 epochs. The loss function was averaged over the 25 samples in each mini-batch and that average loss function was used in the gradient calculation. The gradient was calculated using the parameter-shift rule in Equation (85). To classify a sample in the validation and testing phases of the machine learning process, the continuously valued expectation value output from the network is thresholded. Outputs greater than or equal to 0 are classified as class 0 and outputs less than 0 are classified as class 1.

Using these methods, the network was successfully trained and the results found match the results given in [40]. The loss was recorded for every batch, and the plot of the loss vs. batch is shown below in Figure 18.

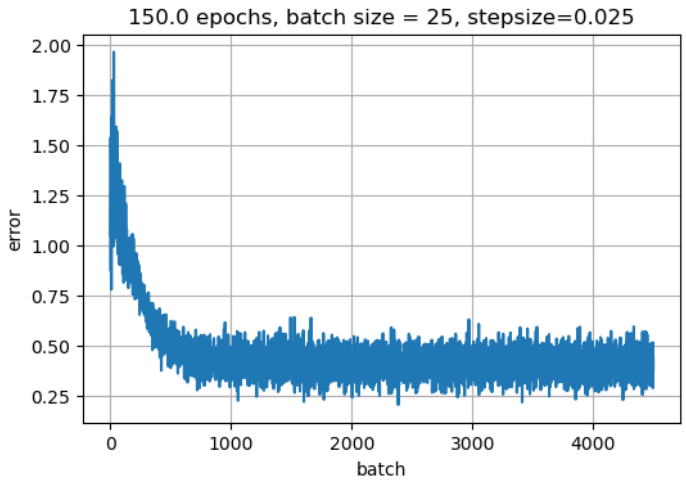


Figure 18: Loss by batch over the training period

The loss decreased to less than 0.5 on average, and plateaued fairly early in the training process. The plot of validation accuracy per batch over the training period also plateaued early in training as well, as shown in Figure 19.

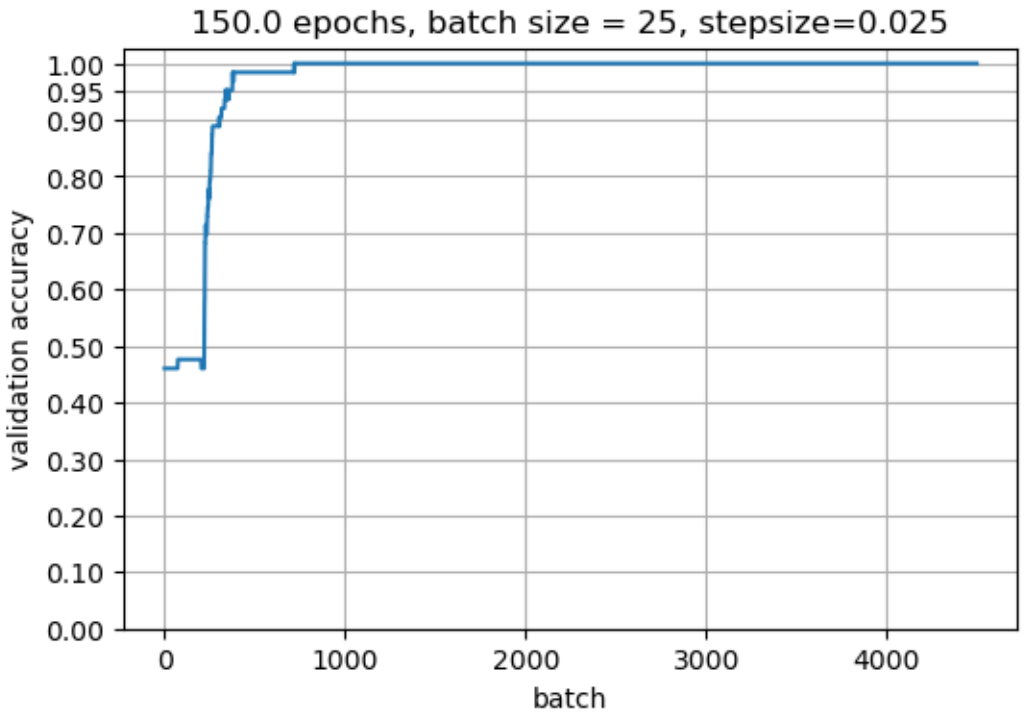


Figure 19: Validation accuracy vs batch over training period

The validation accuracy converges to 100% after about 1000 batches, which is an indicator of good network performance. In testing the network performed very well, correctly classifying 100% of the testing data samples. The correct labels and the classification results from the network on the test set are shown below in Figure 20.

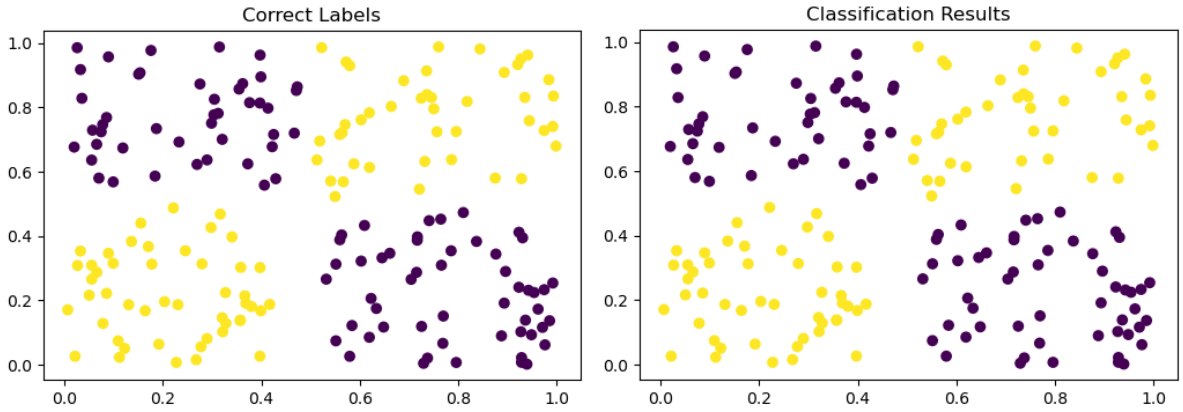


Figure 20: Correct (left) and network assigned (right) labels for test data

The plots are identical, showing that the network assigned the correct label to every testing sample. Furthermore, the parameters used and the values they converged to are in full agreement with the parameters used in [40]. The evolutions of the parameters over the training period are shown below in Figure 21 and Figure 22, where the ideal parameter values from [40] are shown by dashed lines.

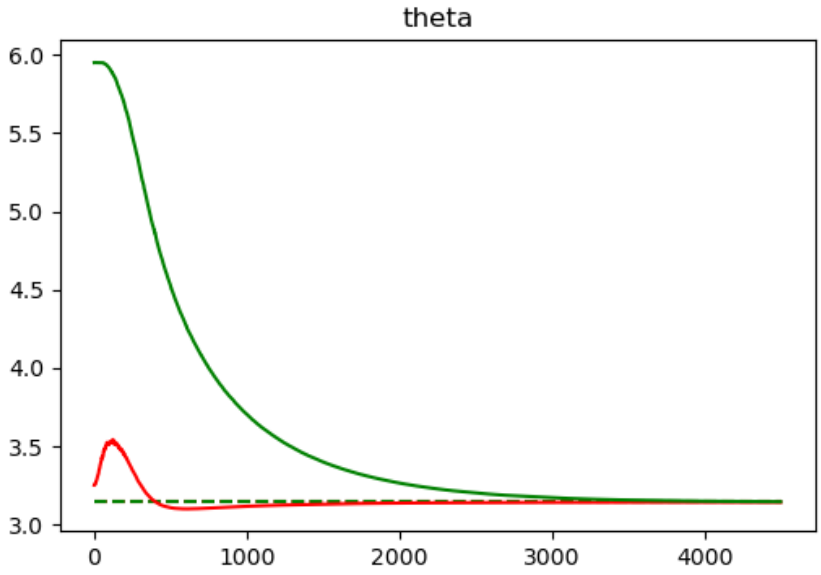


Figure 21: Theta parameter training

In Figure 21, the dashed lines are at exactly π, which correspond to the value used for θ_1 and θ_2 in [40]. The network parameters trained here, shown with solid lines, converge to approximately π. The final values for θ_1 and θ_2 at the end of training were 3.14147 and 3.14479, respectively.

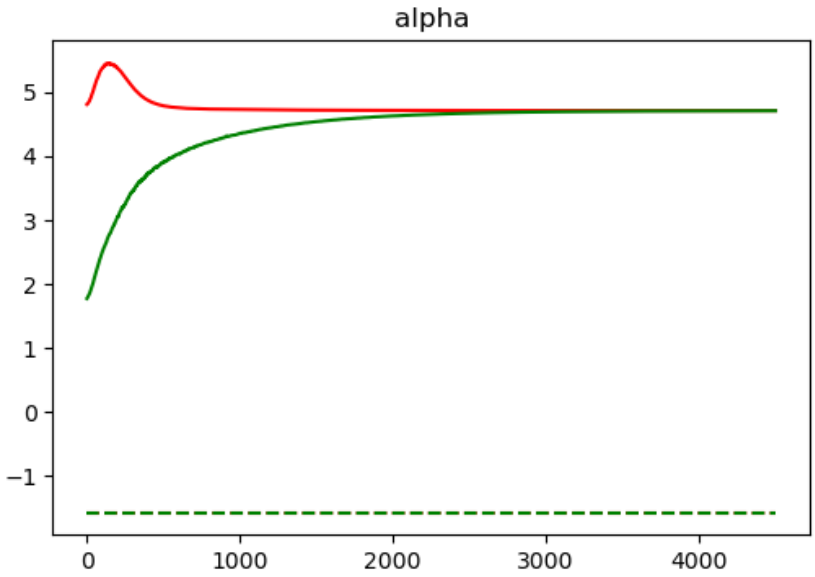


Figure 22: Alpha parameter training

The results for the α parameters are shown in Figure 22. The dashed lines are at exactly $-\pi/2$, which correspond to the values used for α_1 and α_2 in [40]. The network parameters α_1 and α_2 converged to values of 4.71247 and 4.71022. At first it appears that the network parameters are not in agreement as they converged to different values, however these values are approximately equal to $+3\pi/2$. Since rotations wrap from 2π back to 0, a rotation by $+3\pi/2$ is equivalent to a rotation by $-\pi/2$. Thus the circuit trained here is equivalent to the circuit presented in [40]. These results indicate that quantum circuits can be trained (in simulation) using quantum machine learning methods. Since the XOR problem is an example of a problem that can be solved with a single quantum neuron in contrast to a multi-layer classical perceptron, this simulated demonstration highlights some of the potential advantages of quantum computing and quantum machine learning.

CONCLUSIONS

This manuscript introduces the relevant concepts of quantum machine learning, and serves as introductory material. The basic notions of quantum mechanics are described, including quantum phase, superposition, entanglement, and expectations. These are used to introduce quantum gates as fundamental building blocks of the quantum computing framework in comparison with the classical digital logic framework. The basics of classical machine learning are introduced specifically related to deep learning for classification, and are used as a background in order to introduce standard notions in quantum machine learning. Finally these notions are applied to an example problem that highlights some potential advantages of quantum machine learning over its classical counterpart. With the growing capabilities of quantum computers, quantum machine learning holds promise for solving hard problems in a variety of domains, and warrants further investigation into the quantum advantage of quantum machine learning.

REFERENCES

[1] A. Turing, "Intelligent machinery (1948)," *B. Jack Copeland,* p. 395, 2004.

[2] K. Naja, S. F. Yelin and X. Gao, "The development of quantum machine learning," *Harvard Data Science Review,* vol. 4, 2022.

[3] A. Adamatzky, Advances in unconventional computing: Volume 1: Theory, Springer, 2016.

[4] A. Adamatzky, B. D. L. Costello and T. Asai, Reaction-diffusion computers, Elsevier, 2005.

[5] B. J. Shastri, A. N. Tait, T. Ferreira de Lima, W. H. Pernice, H. Bhaskaran, C. D. Wright and P. R. Prucnal, "Photonics for artificial intelligence and neuromorphic computing," *Nature Photonics,* vol. 15, pp. 102-114, 2021.

[6] S. Aaronson, "Introduction to Quantum Information Science," 2018. [Online]. Available: https://www.scottaaronson.com/qclec.pdf. [Accessed 15 Feb. 2023].

[7] Y. Wang, Z. Hu, B. C. Sanders and S. Kais, "Qudits and high-dimensional quantum computing," *Frontiers in Physics,* vol. 8, p. 589504, 2020.

[8] S. Lloyd and S. L. Braunstein, "Quantum computation over continuous variables," *Physical Review Letters,* vol. 82, p. 1784, 1999.

[9] O. Pfister, "Continuous-variable quantum computing in the quantum optical frequency comb," *Journal of Physics B: Atomic, Molecular and Optical Physics,* vol. 1, p. 012001, 2019.

[10] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse Ising model," *Physical Review E,* vol. 58, p. 5355, 1998.

[11] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori and W. D. Oliver, "Perspectives of quantum annealing: Methods and implementation," *Reports on Progress in Physics,* vol. 83, p. 054401, 2020.

[12] J. Biamonte, P. Wittek, N. Pancotti, p. Rebentrost, n. Wiebe and S. Lloyd, "Quantum machine learning," *Nature,* vol. 549, pp. 195-202, 2017.

[13] K. Najafi, S. F. Yelin and X. Gao, "The development of quantum machine learning," *Harvard Data Science Review,* vol. 4, 2022.

[14] "Our quantum computing journey," Google, [Online]. Available: https://quantumai.google/learn/map. [Accessed 15 Feb. 2023].

[15] J. S. Townsend, Quantum Physics: A Fundamental Approach to Modern Physics., Mill Valley, CA: University Sciences Book, 2010.

[16] N. Zettili, Quantum Mechanics: Concepts and Applications 2nd ed., Hoboken: John Wiley & Sons, Inc, 2009.

[17] M. Le Bellac, Quantum Physics., Cambridge: Cambridge University Press, 2006.

[18] M. A. Nielson and I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge: Cambridge University Press, 2010.

[19] P. Kaye, R. Laflamme and M. Mosca, An introduction to quantum computing, Oxford: Oxford University Press, 2007.

[20] J. a. E. R. a. P. A. a. G. T. a. F. M. a. R. O. a. T. K. a. B. R. a. Z. A. a. P. A. a. o. Jumper, "Highly accurate protein structure prediction with AlphaFold," *Nature,* vol. 596, no. 7873, pp. 583-589, 2021.

[21] J. a. S. S. a. H. P. a. R. S. a. C. A. a. M. M. a. K. T. a. H. D. a. L. Z. a. A. K. a. o. Pathak, "Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators," *ArXiv,* no. arXiv:2202.11214, 2022.

[22] OpenAI, "GPT-4 Technical Report," *arXiv,* vol. 2303.08774, 2023.

[23] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, B. Roelogs, B. Sapp, B. White, A. Faust, A. Whiteson, S. Whiteson and others, "Imitation Is Not Enough: Robustifying Imitation with Reinforcement Learning for Challenging Driving Scenarios," *arXiv preprint arXiv:2212.11419,* 2022.

[24] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas and others, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature,* vol. 602, pp. 414--419, 2022.

[25] F. Rosenblatt, "The Perceptron - a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, 1957.

[26] N. Wiener and T. Teichmann, "Nonlinear Problems in Random Theory," 1958.

[27] F. F. Rosenblatt, "Principles of Neurodynamics. Perceptrons and the theory of brain mechanisms," *American Journal of Psychology,* vol. 76, p. 705, 1963.

[28] K. Hornik, M. B. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks,* vol. 2, pp. 359-366, 1989.

[29] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, Cambridge, MA, USA, MIT Press, 1986, pp. 318-362.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation,* vol. 1, pp. 541-551, 1989.

[31] C. M. Bishop, "Nural Networks," in *Pattern Recognition and Machine Learning*, New York, NY, Springer, 2009, pp. 225-290.

[32] V. e. a. Bergholm, PennyLane: Automatic differentiation of hybrid quantum-classical computations, arXiv, 2018.

[33] J. Izaac, "Basic tutorial: qubit rotation," PennyLane, 11 October 2019. [Online]. Available: https://pennylane.ai/qml/demos/tutorial_qubit_rotation.html. [Accessed 2022].

[34] J. Izaac, "Quantum gradients with backpropagation," PennyLane, 11 August 2020. [Online]. Available: https://pennylane.ai/qml/demos/tutorial_backprop.html#quantum-gradients-with-backpropagation. [Accessed 2022].

[35] "Parameter-shift rules," Xanadu, 2022. [Online]. Available: https://pennylane.ai/qml/glossary/parameter_shift.html. [Accessed 2022].

[36] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac and N. Killoran, "Evaluating analytic gradients on quantum hardware," *Physical Review A,* vol. 99, 2019.

[37] K. Mitarai, M. Negoro, M. Kitagawa and K. Fujii, "Quantum Circuit Learning," *Physical Review A,* vol. 98, 2018.

[38] N. Killoran, "The stochastic parameter-shift rule," Xanadu, 25 May 2020. [Online]. Available: https://pennylane.ai/qml/demos/tutorial_stochastic_parameter_shift.html#banchi2020. [Accessed 2022].

[39] L. Banchi and G. E. Crooks, "Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule," *Quantum,* vol. 5, p. 386, 2021.

[40] I. V. Grossu, "Single qubit neural quantum circuit for solving exclusive-or," *MethodsX,* vol. 8, p. 101573, 2021.

[41] M. Tipping, "The Relevance Vector Machine," in *Advances in Neural Information Processing Systems*, 1999.

APPENDIX

A. Binary Dimensionality Reduction Example

In this example we will assume that our data has a starting dimension of 200 and that we are using a quantum computer/simulator that has 16 qubits. For this example we will assume that we are embedding binary values. To convert the high dimensional floating point data to a 16 bit binary vector, an ensemble of weak classifiers will be used.

For the weak classifier, the perceptron mentioned in the main document is used. This works well in the ensemble case as the algorithm can be optimized through direct optimization via

$$\boldsymbol{w} = \boldsymbol{y}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1},$$

(99)

where $\boldsymbol{y}$ is the vector of true labels in $\{-1, 1\}$ and $\boldsymbol{X}$ is the set of all training data. This is the solution to solving the equation $\boldsymbol{y} = \boldsymbol{w}\boldsymbol{X}$ for $\boldsymbol{w}$.

To train an ensemble of perceptrons the training data is split into N sets of equal size, where N is the number of desired bits in the quantum encoding. A perceptron is then trained, using Equation (99), for each of the N sets. The entirety of the data is then passed through each of the N perceptrons. Each -1 is converted to zero, and the N outputs are combined together to form the binary representation for the algorithm comparisons.

B. Proof of the Pauli Commutator identity

Proof of the Pauli Commutator Identity

Ethan N. Evans*

1. Statement and Proof of the Pauli Commutator Identity. Let σ_i be a Pauli matrix, and B any operator. Let $U_i(\theta) := \exp(-i\frac{\theta}{2}\sigma_i)$. Then

$$[\sigma, B] = -i\Big(U_i^\dagger\Big(\frac{\pi}{2}\Big)BU_i\Big(\frac{\pi}{2}\Big) - U_i^\dagger\Big(-\frac{\pi}{2}\Big)BU_i\Big(-\frac{\pi}{2}\Big)\Big) \tag{1.1}$$

Proof. First, note that for Pauli matrix σ_i, $i = 1, 2, 3$, $\sigma_i^\dagger = \sigma_i$ (that is, Pauli matrices are Hermitian). Plugging in the definition of $U_i(\frac{\pi}{2})$ yields

$$[\sigma, B] = -i\Big(\exp\Big(i\frac{\pi}{4}\sigma_i\Big)B\exp\Big(-i\frac{\pi}{4}\sigma_i\Big) - \exp\Big(-i\frac{\pi}{4}\sigma_i\Big)B\exp\Big(i\frac{\pi}{4}\sigma_i\Big)\Big). \tag{1.2}$$

next, we apply the following Pauli matrix identity:

$$e^{ia(\hat{n}\cdot\vec{\sigma})} = I\cos a + i(\hat{n}\cdot\vec{\sigma})\sin a, \quad |\hat{n}| = 1, \tag{1.3}$$

where $\vec{\sigma} = \sigma_x\hat{x} + \sigma_y\hat{y} + \sigma_z\hat{z}$. Put simply, this intermediate identity holds through the expansion of the exponential into an infinite sum, splitting the sum into even and odd exponent parts, and noting that $\sigma^{2p} = I$, $p \in \mathbb{N}_+$, which causes the cosine (composed of the even exponent terms) to only have prefactor Identity, and the sine (composed of the odd exponent terms) to be left with the Pauli matrix prefactor. In our case, $\sigma_i = \hat{n}_i \cdot \vec{\sigma}$, thus

$$\exp\Big(i\frac{\pi}{4}\sigma_i\Big) = I\cos\frac{\pi}{4} + i\sigma_i\sin\frac{\pi}{4} \tag{1.4}$$

Applying this identity, and also the fact that $\cos(-a) = \cos(a)$ and $\sin(-a) = -\sin(a)$, yields

$$\begin{aligned}[\sigma, B] &= -i\Big(\Big(I\cos\frac{\pi}{4} + i\sigma_i\sin\frac{\pi}{4}\Big)B\Big(I\cos\frac{-\pi}{4} + i\sigma_i\sin\frac{-\pi}{4}\Big) \\ &\quad - \Big(I\cos\frac{-\pi}{4} + i\sigma_i\sin\frac{-\pi}{4}\Big)B\Big(I\cos\frac{\pi}{4} + i\sigma_i\sin\frac{\pi}{4}\Big)\Big) \qquad (1.5)\\ &= -i\Big(\Big(I\cos\frac{\pi}{4} + i\sigma_i\sin\frac{\pi}{4}\Big)B\Big(I\cos\frac{\pi}{4} - i\sigma_i\sin\frac{\pi}{4}\Big) \\ &\quad - \Big(I\cos\frac{\pi}{4} - i\sigma_i\sin\frac{\pi}{4}\Big)B\Big(I\cos\frac{\pi}{4} + i\sigma_i\sin\frac{\pi}{4}\Big)\Big) \qquad (1.6)\\ &= -i\Big(\Big(I\frac{\sqrt{2}}{2} + i\sigma_i\frac{\sqrt{2}}{2}\Big)B\Big(I\frac{\sqrt{2}}{2} - i\sigma_i\frac{\sqrt{2}}{2}\Big) - \Big(I\frac{\sqrt{2}}{2} - i\sigma_i\frac{\sqrt{2}}{2}\Big)B\Big(I\frac{\sqrt{2}}{2} + i\sigma_i\frac{\sqrt{2}}{2}\Big)\Big) \qquad (1.7)\\ &= -\frac{i}{2}\Big(B - iB\sigma_i + i\sigma_iB - i^2\sigma_iB\sigma_i - \Big(B + iB\sigma_i - i\sigma_iB - i^2\sigma_iB\sigma_i\Big)\Big) \qquad (1.8)\\ &= -\frac{i}{2}\Big(-2iB\sigma_i + 2i\sigma_iB\Big) = \sigma_iB - B\sigma_i \qquad (1.9)\end{aligned}$$

■

*Naval Surface Warfare Center, Panama City Division (ethan.n.evans.civ@us.navy.mil)

C. Baker-Campbell-Hausdorff Derivation/Proof

Starting with the function $f(\lambda) = e^{\lambda\hat{A}}\hat{B}e^{-\lambda\hat{A}}$, which is the same form as what is typically found in an expectation value calculation, write it as a Taylor series expansion. Taylor series have the form:

$$F(x) = \sum_{n=0}^{\infty} \frac{F^{(n)}(a)(x-a)^n}{n!}$$

(100)

So taking the first derivative of $f(\lambda) = e^{\lambda\hat{A}}\hat{B}e^{-\lambda\hat{A}}$:

$$f'(\lambda) = e^{\lambda\hat{A}}\hat{A}\hat{B}e^{-\lambda\hat{A}} - e^{\lambda\hat{A}}\hat{B}\hat{A}e^{-\lambda\hat{A}} = e^{\lambda\hat{A}}[\hat{A},\hat{B}]e^{-\lambda\hat{A}}$$

(101)

Then evaluating for $\lambda = 0$:

$$f'(0) = e^{0*\hat{A}}\left[\hat{A},\hat{B}\right]e^{-0*\hat{A}} = \left[\hat{A},\hat{B}\right] = [\hat{A},\cdot]^1\hat{B},$$

(102)

where $[\hat{A},\cdot]^1\hat{B}$ is an alternative notation whose usefulness will become apparent shortly. Repeating the above steps for the second derivative gives:

$$f''(\lambda) = e^{\lambda\hat{A}}\hat{A}\left[\hat{A},\hat{B}\right]e^{-\lambda\hat{A}} - e^{\lambda\hat{A}}\left[\hat{A},\hat{B}\right]\hat{A}e^{-\lambda\hat{A}} = e^{\lambda\hat{A}}\left[\hat{A},\left[\hat{A},\hat{B}\right]\right]e^{-\lambda\hat{A}} = e^{\lambda\hat{A}}[\hat{A},\cdot]^2\hat{B}e^{-\lambda\hat{A}}$$

$$f''(0) = e^{0*\hat{A}}\left[\hat{A},\left[\hat{A},\hat{B}\right]\right]e^{-0*\hat{A}} = \left[\hat{A},\left[\hat{A},\hat{B}\right]\right] = [\hat{A},\cdot]^2\hat{B}.$$

(103)

Using the developing pattern, the nth derivative evaluated at zero can be written as:

$$f^{(n)}(0) = [\hat{A},\cdot]^n\hat{B}.$$

(104)

Then writing $f(\lambda)$ as a Taylor series expansion (100) centered at zero (i.e. with $a = 0$) gives:

$$f(\lambda) = e^{\lambda\hat{A}}\hat{B}e^{-\lambda\hat{A}} = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)(\lambda-0)^n}{n!} = \sum_{n=0}^{\infty} \frac{[\hat{A},\cdot]^n\hat{B}(\lambda)^n}{n!}$$

(105)

The final form of the expression looks like the definition of an exponential,

$$e^{\lambda\hat{A}} = \sum_{n=0}^{\infty} \frac{\lambda^n\hat{A}^n}{n!}.$$

(106)

Rewriting $f(\lambda)$ in exponential form yields:

$$f(\lambda) = e^{\lambda\hat{A}}\hat{B}e^{-\lambda\hat{A}} = \sum_{n=0}^{\infty} \frac{[\hat{A},\cdot]^n \hat{B}(\lambda)^n}{n!} = \left(\sum_{n=0}^{\infty} \frac{(\lambda)^n [\hat{A},\cdot]^n}{n!}\right)\hat{B} = e^{\lambda[\hat{A},\cdot]}\hat{B},$$

(107)

which is the Baker-Campbell-Hausdorff identity.

D. Derivation of the Derivative of a Parametric Exponential Operator

Derivation of the Derivative of a Parametric Exponential Operator

Ethan N. Evans*

1. Statement and Derivation of the Identity. Let $Z(\theta)$ be a some operator parameterized by θ. Then

$$\frac{\partial}{\partial\theta}e^{Z(\theta)} = \int_0^1 e^{(1-s)Z(\theta)}\frac{\partial}{\partial\theta}Z(\theta)e^{sZ(\theta)}\mathrm{d}s \tag{1.1}$$

Proof. First, note that $e^{Z(\theta)} = \sum_{n=0}^{\infty}\frac{Z(\theta)^n}{n!}$. So

$$\frac{\partial}{\partial\theta}e^{Z(\theta)} = \sum_{n=0}^{\infty}\frac{1}{n!}\frac{\partial}{\partial\theta}Z(\theta)^n \tag{1.2}$$

$$= \frac{\partial}{\partial\theta}\Big(I + Z(\theta) + \frac{1}{2}Z(\theta)^2 + \dots\Big) \tag{1.3}$$

$$= 0 + \frac{\partial Z(\theta)}{\partial\theta} + \frac{1}{2}\Big(Z(\theta)\frac{\partial Z(\theta)}{\partial\theta} + \frac{\partial Z(\theta)}{\partial\theta}Z(\theta)\Big) +$$
$$+ \frac{1}{6}\Big(\frac{\partial Z(\theta)}{\partial\theta}Z(\theta)^2 + Z(\theta)\frac{\partial Z(\theta)}{\partial\theta}Z(\theta) + Z(\theta)^2\frac{\partial Z(\theta)}{\partial\theta}\Big)$$
$$+ \frac{1}{24}\Big(\frac{\partial Z(\theta)}{\partial\theta}Z(\theta)^3 + Z(\theta)\frac{\partial Z(\theta)}{\partial\theta}Z(\theta)^2 + Z(\theta)^2\frac{\partial Z(\theta)}{\partial\theta}Z(\theta) + Z(\theta)^3\frac{\partial Z(\theta)}{\partial\theta}\Big) + \dots \tag{1.4}$$

$$= I\frac{\partial Z(\theta)}{\partial\theta}I + \Big(\frac{1}{2}Z(\theta) + \frac{1}{6}Z(\theta)^2 + \frac{1}{24}Z(\theta)^3 + \dots\Big)\frac{\partial Z(\theta)}{\partial\theta}I$$
$$+ I\frac{\partial Z(\theta)}{\partial\theta}\Big(\frac{1}{2}Z(\theta) + \frac{1}{6}Z(\theta)^2 + \frac{1}{24}Z(\theta)^3 + \dots\Big)$$
$$+ \Big(\frac{1}{24}Z(\theta)^2 + \dots\Big)\frac{\partial Z(\theta)}{\partial\theta}Z(\theta) + Z(\theta)\frac{\partial Z(\theta)}{\partial\theta}\Big(\frac{1}{24}Z(\theta)^2 + \dots\Big) + \dots \tag{1.5}$$

$$= \sum_{n=0}^{\infty}\sum_{k=0}^{n}\frac{1}{(n+1)!}Z(\theta)^k\frac{\partial Z(\theta)}{\partial\theta}Z(\theta)^{n-k}. \tag{1.6}$$

Now, shifting/swapping the indices using the identity

$$\sum_{n=0}^{\infty}\sum_{k=0}^{n}f_{n,k} = \sum_{k=0}^{\infty}\sum_{n=k}^{\infty}f_{n,k} = \sum_{k=0}^{\infty}\sum_{n=0}^{\infty}f_{n+k,k}, \tag{1.7}$$

we have

$$\frac{\partial}{\partial\theta}e^{Z(\theta)} = \sum_{k=0}^{\infty}\sum_{n=0}^{\infty}\frac{1}{(n+k+1)!}Z(\theta)^k\frac{\partial Z(\theta)}{\partial\theta}Z(\theta)^n. \tag{1.8}$$

*Naval Surface Warfare Center, Panama City Division (ethan.n.evans.civ@us.navy.mil)

Now multiply by $1 = \frac{n!k!}{n!k!}$

$$\frac{\partial}{\partial\theta}e^{Z(\theta)} = \sum_{k=0}^{\infty}\sum_{n=0}^{\infty}\frac{n!k!}{n!k!(n+k+1)!}Z(\theta)^k\frac{\partial Z(\theta)}{\partial\theta}Z(\theta)^n. \tag{1.9}$$

Next, we note that $\frac{n!k!}{(n+k+1)!}$ is the definition of the beta function, which can be represented as an integral via Euler's beta function identity (proof provided after the conclusion of this proof)

$$B(k+1,n+1) := \frac{n!k!}{(n+k+1)!} = \int_0^1 (1-s)^k s^n \mathrm{d}s \tag{1.10}$$

Plugging in Euler's beta function identity yields

$$\frac{\partial}{\partial\theta}e^{Z(\theta)} = \int_0^1 \sum_{k=0}^{\infty}\sum_{n=0}^{\infty}\frac{1}{n!k!}\Big((1-s)Z(\theta)\Big)^k\frac{\partial Z(\theta)}{\partial\theta}\Big(sZ(\theta)\Big)^n\mathrm{d}s \tag{1.11}$$

$$= \int_0^1 \sum_{k=0}^{\infty}\frac{1}{k!}\Big((1-s)Z(\theta)\Big)^k\frac{\partial Z(\theta)}{\partial\theta}\sum_{n=0}^{\infty}\frac{1}{n!}\Big(sZ(\theta)\Big)^n\mathrm{d}s \tag{1.12}$$

$$= \int_0^1 e^{(1-s)Z(\theta)}\frac{\partial Z(\theta)}{\partial\theta}e^{sZ(\theta)}\mathrm{d}s \tag{1.13}$$

■

2. Statement and Proof of Euler's beta function identity.

$$B(k,n) := \frac{(n-1)!(k-1)!}{(n+k-1)!} = \int_0^1 (1-s)^{k-1} s^{n-1} \mathrm{d}s \tag{2.1}$$

Proof. This proof is readily available in numerous online sources. First, by integration-by-parts, one can show that for $m \in \mathbb{N}$

$$\int_0^\infty p^{m-1} e^{-p} \mathrm{d}p = (m-1)! \tag{2.2}$$

Next, consider the product

$$(m-1)!(n-1)! = \int_0^\infty p^{m-1} e^{-p} \mathrm{d}p \times \int_0^\infty q^{n-1} e^{-q} \mathrm{d}q. \tag{2.3}$$

Rewriting it as a double integral over the region $x, y \geq 0$ in the (x,y)-plane yields

$$(m-1)!(n-1)! = 2\int_0^\infty x^{2m-1} e^{-x^2} \mathrm{d}x \times 2\int_0^\infty y^{2n-1} e^{-y^2} \mathrm{d}y. \tag{2.4}$$

Now transform to polar coordinates: $x = r\cos\theta$, $y = r\sin\theta$, and $\mathrm{d}x\mathrm{d}y = r\mathrm{d}r\mathrm{d}\theta$

$$(m-1)!(n-1)! = 4\int_0^\infty r^{2(m+n-1)} e^{-r^2} r\mathrm{d}r \times \int_0^{\pi/2} \sin^{2n-1}\theta \cos^{2m-1}\theta \mathrm{d}\theta \tag{2.5}$$

$$= 2(m+n-1)! \int_0^{\pi/2} \sin^{2n-1}\theta \cos^{2m-1}\theta \mathrm{d}\theta, \tag{2.6}$$

where we have applied the single integer factorial form in (2.2). Thus, we have that

$$\frac{(m-1)!(n-1)!}{(m+n-1)!} = 2\int_0^{\pi/2} \sin^{2n-1}\theta \cos^{2m-1}\theta \mathrm{d}\theta. \tag{2.7}$$

Finally, make the change of variable $t = sin^2\theta$, which implies that $\mathrm{d}t = 2\sin\theta\cos\theta\mathrm{d}\theta$ and $\cos^2\theta = 1 - t$. Re-writing the integral and applying this change of variables yields

$$2\int_0^{\pi/2} \sin^{2n-1}\theta \cos^{2m-1}\theta \mathrm{d}\theta = \int_0^{\pi/2} \sin^{2n-2}\theta \cos^{2m-2}\theta 2\sin\theta\cos\theta \mathrm{d}\theta \tag{2.8}$$

$$= \int_0^1 (1-t)^{n-1} t^{m-1} \mathrm{d}t \tag{2.9}$$

Thus we have concluded that

$$\frac{(m-1)!(n-1)!}{(m+n-1)!} = \int_0^1 (1-t)^{n-1} t^{m-1} \mathrm{d}t \tag{2.10}$$

■

DISTRIBUTION

	Copies
DEFENSE TECHNICAL INFORMATION CENTER ATTN DTIC-0 8725 JOHN J KINGMAN ROAD FORT BELVOIR VA 22060-6218	1
OFFICE OF NAVAL RESEARCH 875 N. RANDOLPH STREET SUITE W100 ARLINGTON, VA 20373-5123	1
NAVAL SURFACE WARFARE CENTER PANAMA CITY DIVISION ATTN CODE E13 110 VERNON AVENUE PANAMA CITY, FL 32407	2
NAVAL SURFACE WARFARE CENTER PANAMA CITY DIVISION ATTN CODE X11 110 VERNON AVENUE PANAMA CITY, FL 32407	2


NAVSEA
WARFARE CENTERS
PANAMA CITY