Quantum Artificial Intelligence - Learning Unitary Transformations

by

Bobak Toussi Kiani

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author . . . .
Department of Mechanical Engineering
May 15, 2020

Certified by . . . .
Seth Lloyd
Professor
Thesis Supervisor

Accepted by . . . .
Nicolas Hadjiconstantinou
Chairman, Department Committee on Graduate Theses

Quantum Artificial Intelligence - Learning Unitary Transformations

by

Bobak Toussi Kiani

Submitted to the Department of Mechanical Engineering
on May 15, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Linear algebra is a simple yet elegant mathematical framework that serves as the mathematical bedrock for many scientific and engineering disciplines. Broadly defined as the study of linear equations represented as vectors and matrices, linear algebra provides a mathematical toolbox for manipulating and controlling many physical systems. For example, linear algebra is central to the modeling of quantum mechanical phenomena and machine learning algorithms.

Within the broad landscape of matrices studied in linear algebra, unitary matrices stand apart for their special properties, namely that they preserve norms and have easy to calculate inverses. Interpreted from an algorithmic or control setting, unitary matrices are used to describe and manipulate many physical systems. Relevant to the current work, unitary matrices are commonly studied in quantum mechanics where they formulate the time evolution of quantum states and in artificial intelligence where they provide a means to construct stable learning algorithms by preserving norms.

One natural question that arises when studying unitary matrices is how difficult it is to learn them. Such a question may arise, for example, when one would like to learn the dynamics of a quantum system or apply unitary transformations to data embedded into a machine learning algorithm. In this thesis, I examine the hardness of learning unitary matrices both in the context of deep learning and quantum computation. This work aims to both advance our general mathematical understanding of unitary matrices and provide a framework for integrating unitary matrices into classical or quantum algorithms.

Different forms of parameterizing unitary matrices, both in the quantum and classical regimes, are compared in this work. In general, experiments show that learning an arbitrary $d \times d$ unitary matrix requires at least d^2 parameters in the learning algorithm regardless of the parameterization considered. In classical (non-quantum) settings, unitary matrices can be constructed by composing products of operators that act on smaller subspaces of the unitary manifold. In the quantum setting, there also exists the possibility of parameterizing unitary matrices in the Hamiltonian setting, where it is shown that repeatedly applying two alternating Hamiltonians is sufficient to learn an arbitrary unitary matrix.

Finally, I discuss applications of this work in quantum and deep learning settings. For near term quantum computers, applying a desired set of gates may not be efficiently possible. Instead, desired unitary matrices can be learned from a given set of available gates (similar to ideas discussed in quantum controls). Understanding the learnability of unitary matrices can also aid efforts to integrate unitary matrices into neural networks and quantum deep learning algorithms. For example, deep learning algorithms implemented in quantum computers may leverage parameterizations discussed here to form layers in a quantum learning architecture.

Thesis Supervisor: Seth Lloyd
Title: Professor

Acknowledgments

This work would not have been possible without the help and support of my friends, colleagues, and family. Thanking everyone who has helped me along the way would be a monumental task, but I will try to do my best in this brief section.

Firstly, I would like to thank my family. My mom and dad for encouraging me to pursue a career in science and staying with me throughout the flow of my career. My twin brother, Mehrdad, who charted a path in academia before me, and has shown that an academic career can be challenging, enjoyable, and enticing. To my grandparents, both here and in Iran, for their love and support. Also to my my uncles and aunts who give me the energy to work hard. To my cousins: Nima, Ava, Borna, Kasra, Mana, Parsa, and Niki. Despite being the youngest members of the family, you all have been very supportive. Thanks for listening to my awful jokes and helping me out along the way.

I also thank my advisor, Seth Lloyd, for mentoring me during the course of my Masters studies. My progress in graduate school has been much smoother having you as a mentor. Your curiosity for science and your initiative for tackling challenging problems is infectious. To my post-docs, Milad, David, and Giacomo, you have been wonderfully helpful, both as colleages and mentors. To my fellow graduate students, Reevu, Ryuji, Can, Julian, and Lara, thanks for putting up with me and working with me on these problems. This work simply would not be possible without you.

I must give a special thanks to Julian Heidenreich who worked with me on a class project that now forms chapter 3 of this thesis.

To my friends both inside and outside of MIT, I thank you as well. Thanks for keeping me honest and listening to me talk, perhaps ramble, about my work. It feels great knowing there are people outside academia who care about my research enough to entertain me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Linear algebra is among the simplest frameworks for mathematical analysis, and yet, it applies to a wide range of phenomena in science and engineering. Due to its simplicity and widespread applicability, arguably no topic within mathematics has been as extensively studied as linear algebra. Within this richly studied field lies a diverse set of tools commonly used by scientists, engineers, and other practitioners.

A simple definition of linear algebra is the study of linear transformations. In most settings, these linear transformations are typically represented as matrices that act on vectors. In more mathematical language, matrices are linear maps between vector spaces. By classifying these matrices, linear algebra can be segmented into different groups of matrices which each have different properties. One particular group or segment that is the focus of this thesis is the unitary group of matrices. This group of matrices consists of the set of linear transformations that preserve the norms (or lengths) of vectors. Though this property may not appear special at first glance, it is essential to explaining many physical phenomenon and designing systems that preserve stability. The property of "unitarity" has a precise and unique mathematical definition, but before diving into any mathematical formalism, examples of transformations that can be represented as unitary matrices are given below to aid the reader's understanding.

- Transformations consisting of rotations do not change the length or norm of a vector and are thus unitary transformations that can be written as unitary matrices.

- Similarly, transformations consisting of reflections around an axis passing through the origin do not change the length or norm of a vector and can also be written as unitary matrices.

- In quantum mechanics, norms of state vectors have a probabilistic interpretation and thus, unitary matrices represent the set of transformations that preserve basic rules of probabilities (*e.g.,* that probability of one of a complete set of events happening sums to one). In other words, preserving the norm is equivalent to preserving the rules of probability distributions in quantum mechanics.

One natural focus of research in the study of unitary matrices is the challenges associated with learning them. In this setting, machine learning algorithms are tasked with learning unitary transformations or various features associated with unitary transformations. In some cases, algorithms learn by observing data related to or derived from a target unitary transformation (*e.g.* by observing a set of input and output vectors that are generated by applying the target unitary matrix to input vectors). In other cases, algorithms learn by directly observing entries of the target unitary matrix.

Designing systems to learn and construct unitary matrices presents unique challenges. The field of machine learning provides a rich set of tools to perform machine learning, but it is unclear how successful those methods are when applied to learning the specific subset of unitary matrices. In fact, many of the common methods used in machine learning often change learning systems in ways that do not preserve unitarity; *e.g.,* algorithms often perform updates to matrix elements which typically do not preserve unitarity.

In this work, we develop methods to learn unitary matrices and analyze their success. The algorithms and methods discussed in this thesis provide a broad framework for analysis and implementation of algorithms for learning unitary matrices.

For example, algorithms here are applicable to problems arising in fields as diverse as quantum controls and deep learning (see applications).

Algorithms for learning unitary matrices are provided in both the setting of classical computation and quantum computation. This is, in the author's understanding, the first in-depth comprehensive review of the various methods to parameterize and learn unitary matrices in both the context of machine learning and quantum computation. Various parameterizations of unitary matrices are proposed and new algorithms are provided in both the quantum and classical contexts for learning arbitrary unitary transformations. Though the algorithms discussed here are implemented on randomly generated data sets, they can be embedded into real-world quantum or classical machine learning algorithms in straightforward fashion. Such practical applications are discussed in the introduction and conclusion.

1.1 Practical Motivation

As a topic of mathematics, analyzing the "learnability" of unitary transformations is useful alone. However, this work is mostly motivated by two practical applications which both would benefit from advances in methods to learn unitary matrices.

First, in the context of quantum computation, researchers are interested in methods that can be used to control quantum systems and develop useful quantum algorithms. Unitary matrices are the means by which quantum mechanical states transform (see next section). Thus, controlling or manipulating a quantum system in turn requires controlling or manipulating unitary matrices. Algorithms for learning unitary matrices provide a direct means to manipulate these quantum systems for the purpose of quantum control or quantum computation. Examples of applications include:

- Parameterizations used to learn an arbitrary matrix can also be implemented as a learning algorithm in a quantum computer.
- Quantum operations in quantum computers are often noisy. Unitary matrix

learning can aid in finding methods to perform quantum operations in a manner that provides more fidelity.

- Methods to learn unitary matrices can also provide a means to implement arbitrary operations on a quantum computer with a structured algorithm. For example, if any arbitrary operation can be learned using a simple learning algorithm, then that learning algorithm also provides a means to implement arbitrary gates.

Second, this work is motivated by recent interest in the deep learning community to integrate unitary matrices into learning algorithms. Intuitively, deep learning algorithms perform "learning" by processing input data through a series of transformations. These transformations may not, in general, be stable – *e.g.,* see section 1.4.1 on exploding/vanishing gradients in recurrent neural networks. Unitary matrices are, in a sense, naturally stable since they preserve norms and future work can leverage this natural stability to build improved deep learning algorithms. Furthermore, interest in implementing deep learning algorithms in quantum computation has also grown in recent years. Since quantum computers perform unitary operations, this thesis can help inform work in the topic of quantum deep learning as well.

1.2 Unitary Matrices - Mathematical Background

A unitary matrix A is often defined as a complex square matrix whose inverse is equal to its conjugate transpose $A^\dagger$ where the dagger (†) is used to indicate the conjugate transpose. Equivalently, a complex square matrix A is also unitary if any of the following conditions are met [75]:

- The inverse of A is its conjugate transpose $A^\dagger$ (as stated before)
- A has orthonormal columns
- A has orthonormal rows

- The $N \times N$ matrix A is normal and has N eigenvalues $\{\lambda_1, \lambda_2, ..., \lambda_N\}$ all with magnitude equal to 1 ($|\lambda_i| = 1$ for all i)

- A preserves norm: $\|Az\| = \|z\|$ where z is any vector and $\|\cdot\|$ is the standard 2-Norm of a vector space (A is an isometry)

Note, that for the remainder of this thesis, we will assume that all matrices are complex unless otherwise stated. All the above conditions can also be considered as properties of unitary matrices.

Well-known examples of unitary matrices include the Pauli Matrices ($\sigma_{\mathrm{x}}, \sigma_{\mathrm{y}}, \sigma_{\mathrm{z}}$), which are a set of 2×2 unitary matrices that account for the interactions of the spin of a particle with a magnetic field in quantum mechanics. The Pauli Matrices are also Hermitian matrices (equal to their own conjugate transpose). It is straightforward to evaluate the above conditions on the Pauli matrices shown below.

$$\boldsymbol{\sigma}_{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{\sigma}_{\mathbf{y}} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \boldsymbol{\sigma}_{\mathbf{z}} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1.1}$$

1.3 Role of Unitary Matrices in Quantum Mechanics

This short section provides a brief overview of the role of unitary matrices in quantum mechanics. Though this section does not assume any prior knowledge of quantum mechanics, the material covered here only serves as a limited introduction and motivation to quantum mechanics. A complete introduction to the material would require external reference, and the reader is referred to a number of excellent textbooks that can provide a more rigorous and detailed background of quantum mechanics [30, 56, 71].

In quantum mechanics, the state of a particle or system is mathematically given by its wavefunction, often written as Ψ or ψ. The wavefunction is complex valued and can be written as a function of a continuous variable (e.g. time or space) or a discrete variable (e.g. spin). For example, in the discrete case of a spin-$1/2$ particle, the wavefunction can be written as a superposition of two spin states: spin up ($|\uparrow\rangle$) and spin down ($|\downarrow\rangle$).

Given a particle or system, the complete set of all states of the system form a vector space or more specifically, a Hilbert space. In the usual Schrödinger presentation of quantum mechanics, the Hilbert space of wavefunctions is infinite dimensional, and the unitary transformations that govern quantum time evolution are operators rather than matrices. Infinite dimensional Hilbert spaces are rich mathematical structures that require complex and subtle treatment. By contrast, the mathematics of finite-dimensional Hilbert spaces is more straight forward. Fortunately, any quantum system that occupies bounded space and has bounded energy occupies only a finite-dimensional subspace of Hilbert space. In this thesis, therefore, we restrict our attention to finite dimensional Hilbert spaces. The problem of learning unitary operators on infinite dimensional Hilbert spaces raises interesting questions which lie outside of the scope of the present work.

In the case of the spin-$^1\!/_2$, this is a two dimensional Hilbert space where the spin up ($|\uparrow\rangle$) and spin down ($|\downarrow\rangle$) states are commonly chosen as a basis.

$$|\uparrow\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \, |\downarrow\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1.2}$$

Given a complete basis, any vector $|\Psi\rangle$ in the Hilbert space can be written as a linear combination of the basis elements. For example,

$$|\Psi\rangle = \alpha\,|\uparrow\rangle + \beta\,|\downarrow\rangle \tag{1.3}$$

In the above, we have used bra-ket notation. Here, a ket ($|\cdot\rangle$) is used to indicate a vector and a bra ($\langle\cdot|$) is used to indicate the conjugate transpose of a vector (in the dual space). This notation serves only to simplify the equations used in quantum mechanics. We can use bra-ket notation to determine the projection of a wavefunction onto a specific state. For example,

$$\langle\uparrow| = \begin{bmatrix} 1 & 0 \end{bmatrix}, \, |\Psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \tag{1.4}$$

$$\langle\uparrow|\Psi\rangle = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \alpha \tag{1.5}$$

Given a wavefunction, the probability of measuring a particle in a given state is equal to the squared amplitude of the wavefunction projected onto the given state. Continuing the prior example,

$$P(|\uparrow\rangle) = |\langle\uparrow|\Psi\rangle|^2 = \alpha^*\alpha \tag{1.6}$$

where α^* denotes the conjugate transpose of α.

Since the above provides a probabilistic interpretation of the wavefunction, the probabilities must be properly normalized. Namely, for a probability measure to be complete, it must sum to one. This implies that the sum of the probabilities of measuring a wavefunction in the states of a complete basis must sum to one. For our above example which has two basis states, this means that the probability of measuring our wavefunction in the $|\uparrow\rangle$ state plus the $|\downarrow\rangle$ state must sum to one. Equivalently, the 2-norm of a wavefunction must be equal to one in order to conform to rules of probability.

$$\||\Psi\rangle\|_2 = 1 \tag{1.7}$$

Any actions that change the wavefunction must also ensure that probabilities are properly normalized. In other words, the set of operations that can be applied to a wavefunction must hold the 2-norm invariant. For our discrete Hilbert space, this set of operations is precisely the space of unitary matrices. Herein lies the key fact: transformations of a quantum state can be written as unitary transformations on a vector space.

1.3.1 Connection Between Hamiltonians and Unitary Matrices

The time evolution of a quantum system is governed by the Schrödinger equation [30, 71].

$$i\hbar\frac{d}{dt}|\Psi(t)\rangle = H|\Psi(t)\rangle \tag{1.8}$$

where $\hbar = \frac{h}{2\pi}$ is the reduced Planck constant, $\Psi(t)$ is the wavefunction that can vary over time, and H is the Hamiltonian operator.

When the Hamiltonian operator does not depend on time, there exist stationary states corresponding to the eigenfunctions or eigenvectors of the Hamiltonian.

$$H|\Psi(t)\rangle = E|\Psi(t)\rangle \tag{1.9}$$

where E is the eigenvalue of the stationary state. For these stationary states, solving the Schrödinger equation is straightforward. Namely, if $|\Psi(0)\rangle$ is our initial state at $t = 0$, then we obtain the following time evolution.

$$|\Psi(t)\rangle = e^{-\,iEt/\hbar}|\Psi(0)\rangle \tag{1.10}$$

For discrete Hilbert spaces, the Hamiltonian can be represented as a complex valued Hermitian matrix. In this case, the time evolution can be generalized beyond stationary states.

$$|\Psi(t)\rangle = e^{-iHt}|\Psi(0)\rangle \tag{1.11}$$

where for simplicity, we have subsumed the constant $\hbar$ into the Hamiltonian. As long as the Hamiltonian matrix is Hermitian, the time evolution of the above system is equivalent to a unitary transformation, i.e. e^{-iHt} will be evaluated as a unitary matrix.

1.3.2 Quantum Computation

In classical computation, gates are performed on discrete, often binary, states. In quantum computation, states are elements of the Hilbert space. Most commonly, quantum states are constructed as a register of qubits (individual 2-dimensional quantum states) in a tensor product structure [56]. Here, a quantum bit or qubit is a particle in a superposition of two basis states (e.g. $|\uparrow\rangle$ or $|\downarrow\rangle$). Gates applied to a quantum register of many qubits are unitary transformations. These gates come in a variety of forms. The Pauli matrices, introduced earlier are examples of quantum gates. Gates can also be Hamiltonian transformations as discussed in the prior section.

A number of efficient quantum algorithms have been developed that offer exponential speed-up relative to their classical counterpart [56]. However, for these quantum algorithms to be performed successfully, quantum computers need to perform with high fidelity (low error) on large registers of qubits. Quantum computers accessible in the near future are commonly termed noisy Intermediate-Scale Quantum (NISQ) technologies since they are limited in their reliability [60]. The set of operations or gates that are practically available for current quantum computers are often limited. Experimental operation of a gate may be subject to large errors [29, 60]. Thus, better understanding how to parameterize and implement unitary matrices can benefit work in error suppression for quantum computers.

For a more in-depth overview of quantum computation, the reader is referred to [56].

1.4 Neural Networks

Over the past decade, neural networks have performed very well in a variety of different deep learning applications including image recognition, financial modeling, and natural language processing (NLP) [4, 28, 47]. Progress in the field of deep learning has coincided with deeper networks, which learn relationships in the data through a large number of "learning" layers [48, 74]. However, one of the major challenges associated with training deep neural networks is the issue of vanishing and exploding gradients, which arise when dominant eigenvalues raised to large powers can rapidly grow or vanish [3, 57]. In recurrent neural networks (RNN), where the output of a neuron is fed back as an input to the neuron itself, exploding or vanishing gradients are especially problematic. Long data entries passed into the network sequentially amplify gradients of matrices that have eigenvalues with magnitude less than or greater than one [3, 57].

Prior research has proposed various architectures to address this vanishing and exploding gradient problem. These include Long Short Term Memory (LSTM) networks [34], Gated Recurrent Units (GRUs) [11] and Bidirectional RNNs (BRNN) [5]. With these architectures, it is possible to successfully train deeper networks but each still produces a deep architecture potentially susceptible to the intrinsic problem of vanishing and exploding gradients.

A more direct method to overcome issues with vanishing and exploding gradients is by integrating unitary matrices into neural networks [2,35,38]. This approach leverages the fact that the eigenvalues of unitary complex-valued matrices have absolute values of unity. Thus, the magnitude of their eigenvalues cannot grow or shrink when raised to a power.

1.4.1 More on Recurrent Neural Networks (RNN)

A traditional (not recurrent) feedforward neural network processes an input through a series of linear and/or nonlinear transformations (hidden layers) up until an output layer. Feedforward neural networks do not contain any cycles - i.e. information from one layer is never fed back into that layer or prior layers. A feedforward deep neural network with L hidden layers, activation function τ, input x in $\mathbb{R}^{n_{in}}$ and output y in $\mathbb{R}^{n_{out}}$ can be recursively defined by

$$
\begin{aligned}
\phi^{(1)} &= \tau\left(W^{(1)}x + b^{(1)}\right) \\
\phi^{(l)} &= \tau\left(W^{(l)}\phi^{(l-1)} + b^{(l)}\right) \\
y &= W^{(L+1)}\phi^{(L)} + b^{(L+1)}
\end{aligned}
$$

where $\phi^{(l)}$, $b^{(l)} \in \mathbb{R}^{n_l}$, $W^{(l)}$ is an $n_l \times n_{l-1}$ matrix, $n_0 = n_i n$ and $n_{L+1} = n_{out}$.

This is contrasted with the case of a simple recurrent neural network (RNN) where inputs are passed into the network sequentially and hidden layers can also access their prior states in performing calculations. To describe an RNN mathematically, we include a time dimension t as a subscript to index the value of an input or hidden state over time. A simple RNN with L hidden layers, activation function τ, input x_t at time t in $\mathbb{R}^{n_{in}}$ and output y_t at time t in $\mathbb{R}^{n_{out}}$ can be recursively defined by

$$
\begin{aligned}
\phi_t^{(1)} &= \tau\left(W^{(1)}x_t + U^{(1)}\phi_{t-1}^{(1)} + b^{(1)}\right) \\
\phi_t^{(l)} &= \tau\left(W^{(l)}\phi_t^{(l-1)} + U^{(l)}\phi_{t-1}^{(l)} + b^{(l)}\right) \\
y_t &= W^{(L+1)}\phi_t^{(L)} + b^{(L+1)}
\end{aligned}
$$

where $\phi_t^{(l)}$, $b^{(l)} \in \mathbb{R}^{n_l}$, $W^{(l)}$ is an $n_l \times n_{l-1}$ matrix, $U^{(l)}$ is an $n_l \times n_l$ matrix, $n_0 = n_{in}$ and $n_{L+1} = n_{out}$.

To perform learning using a simple RNN, one must setup a cost function C and compute the derivative of the cost function with respect to the entries of weight matrices $\frac{\partial C}{\partial W_{ij}^{(l)}}$. Computing $\frac{\partial C}{\partial W_{ij}^{(l)}}$ requires first computing $\frac{\partial C}{\partial \phi_t^{(l)}}$ [38]

$$
\begin{aligned}
\frac{\partial C}{\partial \phi_t^{(l)}} &= \frac{\partial C}{\partial \phi_T^{(l)}} \frac{\partial \phi_T^{(l)}}{\partial \phi_t^{(l)}} \\
&= \frac{\partial C}{\partial \phi_T^{(l)}} \prod_{k=t}^{T-1} \frac{\partial \phi_{k+1}^{(l)}}{\partial \phi_k^{(l)}} \\
&= \frac{\partial C}{\partial \phi_T^{(l)}} \prod_{k=t}^{T-1} D_k^{(l)} W^{(l)}
\end{aligned} \tag{1.12}
$$

where $D_k^{(l)} = \operatorname{diag}\left\{ \tau' \left(W^{(l)} \phi_t^{(l-1)} + U^{(l)} \phi_{t-1}^{(l)} + b^{(l)} \right) \right\}$ is the Jacobian matrix of the nonlinear activation τ [38]. Given the above, note that the matrix $W^{(l)}$ is applied repetitively, potentially to large powers when T is large. This causes exploding or vanishing gradients if the singular values of $W^{(l)}$ are large or small respectively. Furthermore, this problem worsens as T grows, so learning long range dependencies in a dataset is especially challenging when facing vanishing or exploding gradients.

Here, the use of unitary matrices is readily apparent. Unitary matrices do not exhibit vanishing or exploding gradients since all of their singular values are equal to unity. Thus, constraining the weight matrices $W^{(l)}$ to be unitary is a potential solution to the problem of vanishing or exploding gradients. In practice, enforcing this unitary constraint is challenging since optimization can no longer be performed on the individual entries of the matrix. Much prior research has attempted to integrate unitary matrices into RNNs [2, 38, 54, 77]. The findings from this thesis can help further inform future work into integrating unitary matrices into neural networks.

Chapter 2

Theoretical Overview

2.1 Challenges of Learning Unitary Matrices

To understand the hardness of learning unitary matrices, it is important to provide a brief overview of the unitary group. The unitary group $U(n)$ consists of the set of all $n \times n$ unitary matrices. $U(n)$ forms a Lie group of dimension n^2 and its Lie algebra $u(n)$ consists of the set of $n \times n$ skew Hermitian matrices [31]. Shakespeare once said "the lunatic, the lover, and the poet are of imagination all compact" [73]. Similarly, the space of $n \times n$ unitary matrices is also compact; where in this sense, we mean $U(n)$ is closed and bounded. Furthermore, $U(n)$ is connected, meaning that for any $M_a, M_b \in U(n)$, there exists a continuous path $A(t)$ for $a \leq t \leq b$ such that $A(a) = M_a$ and $A(b) = M_b$. The fact that $U(n)$ is compact and connected gives one hope that a target unitary matrix can be learned by constructing a path to go from a starting unitary matrix to the target unitary matrix. Note, however, that the existence of a continuous path does not imply that it is easy to find that path.

An $n \times n$ unitary matrix contains n^2 parameters. Thus, to learn a desired unitary matrix to a desired accuracy, simple parameter counting would imply that at least d^2 parameters are required in a learning model to learn an arbitrary unitary matrix.

An $n \times n$ unitary matrix also, by definition, has n eigenvalues all of magnitude one. In a sense, this implies that each eigenvalue is equally important when trying to learn an arbitrary unitary matrix. To highlight this in more detail, consider the following problem: our goal is to construct a unitary matrix A that is close to a target unitary matrix B in the Frobenius norm by using a learning model. Mathematically, we would like to minimize a loss function L.

$$\text{Goal: minimize } L = \|A - B\|_2 \tag{2.1}$$

where the Frobenius norm is defined as the square root of the sum of the squares of a matrix's singular values σ_i:

$$\|M\|_2 = \sqrt{\sum_{i=1}^{n} \sigma_i(M)^2} \tag{2.2}$$

The Frobenius norm of an $n \times n$ unitary matrix is thus $\sqrt{n}$. If the eigenvectors of A and B are initially randomly selected, then at initialization, $\|A - B\|_2 \approx \sqrt{2n}$. Assume our learning model is not perfect and can only fit up to $m < n$ of the eigenvalues and eigenvectors of the target matrix B. This would imply that there remain $n - m$ eigenvalues left to fully learn the unitary matrix and the loss achieved is $L \approx \sqrt{2(n-m)}$.

However, typical matrices are not unitary and their singular values range in magnitude. Thus, a learning algorithm can optimally learn the largest eigenvalues first to reduce the loss fastest. Furthermore, it is often the case that matrices considered in the real world have singular values that are zero or close to zero. In these cases, many of the parameters of a matrix can be effectively ignored. Unitary matrices by definition have no null space, and it is thus impossible to "ignore" any eigenvalues or eigenvectors.

2.2 Parameterizing Unitary Matrix Deterministically Using Givens Rotations

Related to the task of learning unitary matrices is the task of factoring a unitary matrix into a sequence of parameterized components. Many methods exist to perform a decomposition or factorization of a given unitary transformation [14, 19]. Notably, these include factorizations of unitary matrices into diagonal and orthogonal matrices [17], recursive methods to parameterize unitary matrices [13, 36], and factorizations of unitary matrices into Givens matrices [23].

As an example, we show how a unitary matrix can be factorized into parameterized Givens rotations using a method given by [23]. Given the values of the entries of a unitary matrix, the factorization described herein provides a deterministic and precise method to reconstruct any unitary with d^2 parameters. Specifically, any unitary matrix can be reconstructed via a sequence of Givens Rotations that each act on two dimensional subspaces.

Givens rotations are rotations in a plane spanned by two coordinates axes. A single givens rotation $G(\theta, \phi)$ acts on a two dimensional subspace and contains two parameters $\theta, \phi \in \mathbb{R}$.

$$G(\theta, \phi) = \begin{bmatrix} \cos(\theta)\, e^{i\phi} & -\sin(\theta)\, e^{-i\phi} \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{2.3}$$

Note, that this Givens matrices used here have a complex component and thus, they differ from the standard Givens matrix form which is an orthogonal matrix. Givens matrices can act on a two dimensional subspace of a larger matrix spanned by the i-th and j-th basis components. We use the notation $G_{i,j}(\theta, \phi)$ to indicate that the Givens rotation acts on the (i, j) plane and place the entries of the Givens matrix on the i-th and j-th rows and columns.

$$G_{i,j}(\theta,\phi) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & \cos(\theta)\, e^{i\phi} & \dots & -\sin(\theta)\, e^{-i\phi} & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & \sin(\theta) & \dots & \cos(\theta) & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}. \tag{2.4}$$

Our goal is to factorize a matrix $T \in U(d)$ with a sequence $G_{i_N,j_N}(\theta_N,\phi_N)\dots G_{i_2,j_2}(\theta_2,\phi_2)G_{i_1,j_1}(\theta_1,\phi_1)$ of length $N = d^2/2$ Givens rotation matrices. If the factorization is successful, $G_{i_N,j_N}(\theta_N,\phi_N)\dots G_{i_2,j_2}(\theta_2,\phi_2)G_{i_1,j_1}(\theta_1,\phi_1)T^\dagger = I^*$, where as a reminder, $T^\dagger$ is the conjugate transpose (inverse) of T. Note, that the matrix I^* is a matrix containing complex numbers on the diagonal all of magnitude unity – one can add a final diagonal matrix to the sequence to set all the diagonal entries to 1. Knowing this, the goal of our factorization is to find Givens matrices that zero-out the off-diagonal components of our sequence until we arrive at a matrix that has all zero entries on the off-diagonals and is thus a diagonal matrix.

We choose the parameters of our Givens matrix $\{i, j, \theta, \phi\}$ to strategically zero out an off-diagonal component of T. Specifically, assume we would like to zero out the (i', j') entry of a matrix M. Without loss of generality, we assume that the (i', j') entry is the top right entry of our matrix M, i.e. $j > i$.

$$M_{i',j'} = \begin{bmatrix} a_{11} + ib_{11} & a_{12} + ib_{12} \\ a_{21} + ib_{21} & a_{22} + ib_{22} \end{bmatrix} \tag{2.5}$$

We construct a Givens matrix acting on the (i', j') plane setting the parameters θ and ϕ such that we obtain a matrix of the following form. Note, that the Givens matrix only acts on the (i', j') so we only consider entries on that plane in the calculations below.

$$G_{i',j'}(\theta',\phi')M_{i',j'} = \begin{bmatrix} * & 0 \\ * & * \end{bmatrix} \tag{2.6}$$

In other words, we solve for θ' and ϕ' such that

$$\begin{bmatrix} \cos(\theta)\,e^{i\phi} & -\sin(\theta)\,e^{-i\phi} \end{bmatrix} \begin{bmatrix} a_{12} + ib_{12} \\ a_{22} + ib_{22} \end{bmatrix} = 0 \tag{2.7}$$

Using the above, we eliminate the off-diagonal entries moving from the top-right non-zero entry and moving down, proceeding in a zig-zag fashion. E.g. for a 4×4 matrix, the order of elimination is below:

$$\begin{bmatrix} * & 6 & 4 & 1 \\ * & * & 5 & 2 \\ * & * & * & 3 \\ * & * & * & * \end{bmatrix} \tag{2.8}$$

Conforming to the above order, once a matrix element is zeroed-out, it remains zeroed- out. For a $d \times d$ matrix, this requires $d(d-1)/2$ applications of Givens matrices. Since each Givens matrix has 2 parameters, this reconstruction requires d^2 parameters in total (as expected). As stated earlier, the result after the application of Givens matrices is a matrix with complex numbers on the diagonals all of magnitude one. To set the diagonal elements exactly to one, we can apply a final diagonal matrix set to the negative phase at each diagonal entry to cancel out phases.

2.3 Relation to Quantum Controls

Methods of learning unitary matrices are related to problems studied in quantum controls, where researchers aim to "steer" quantum systems using a set of control Hamiltonians [7, 12, 14, 18, 27, 41, 46, 64, 76]. Written in a general form, one may aim to control a wavefunction Ψ using controls $\{u_1(t), u_2(t), \ldots, u_n(t)\}$ for a system undergoing Hamiltonian evolution as below [6, 14, 42, 62, 63].

$$i\frac{d\Psi(t)}{dt} = (H_0 + H_1u_1(t) + H_2u_2(t) + \cdots + H_nu_n(t))\Psi(t) \tag{2.9}$$

where H_0 is a drift Hamiltonian (uncontrolled) and $H_1, H_2, \ldots, H_n$ are the control Hamiltonians.

Quantum systems which can be transformed from any given state to a desired final state are termed controllable [6, 16]. Criteria for controllability have been provided in many cases for finite dimensional quantum systems [27, 63, 79].

Notably, Rabitz et al. [63] have shown that quantum control landscapes for a broad class of physical systems are trap free - *i.e.,* free of points in the control landscape where applying any set of controls no longer increase fidelity. By definition, these systems are controllable; however, it is often challenging to define how much resources (*e.g.,* time or number of parameters) are required to fully control a system [6,15,45]. In Chapter 4, we discuss a simple method of learning unitary matrices that is motivated by results from quantum controls. Namely, we parameterize unitaries by applying two random Hamiltonians in an alternating fashion and show that this simple method can effectively learn a desired unitary matrix. Furthermore, we show that learning can be achieved even when this algorithm is given the minimum number of parameters (resources) required to learn an arbitrary unitary.

2.4 Choice of Loss Function

In a sense, learning a unitary matrix is analogous to developing methods to approximate a desired function. To effectively learn unitary matrices, one must choose a suitable loss function for performing optimization and quantifying how well one has approximated or learned a target function. There are typically two important factors to consider in choosing an appropriate loss function. First, the loss function must reflect the reality of the problem at hand, serving as a success metric for a machine learning practitioner. A good loss function should be minimized only when learning has been successfully performed and should indicate how much more learning needs to be done when the loss function is not minimized. Second, and especially important here, the loss function should provide an efficient means of performing optimization. In this thesis which focuses on gradient based optimization methods, this implies that gradients of the loss function with respect to parameters should be efficiently calculable.

There are many reasonable choices of a loss function, and the optimal choice depends on the specifics of the learning problem. Consider the general problem of learning a target unitary U_t. When given a list of input and output vectors generated from that target, one useful loss function compares how well one matches the output vectors. Namely, assume we are given a list of K input and output vectors: $\{v_1, v_2, \ldots, v_K\}$ and $\{U_t v_1, U_t v_2, \ldots, U_t v_K\}$. We would like to construct a matrix $\tilde{U}$ that is close to U_t. A common loss function $\hat{L}$ in this case is that associated with our "training set" of input and output vectors:

$$\hat{L} = \frac{1}{K} \sum_{i=1}^{K} \|U_t v_i - \tilde{U} v_i\|^2 \tag{2.10}$$

There is also the loss function L which compares U_t to $\tilde{U}$ directly, agnostic of any set of vector inputs or outputs.

$$L = \|U_t - \tilde{U}\|^2 \tag{2.11}$$

Many choices exist for the above matrix norm; in this thesis, we use the Frobenius norm since it comes in a closed form that provides an efficient means of calculating gradients.

$$\|U_t - \tilde{U}\|_2^2 = \mathrm{Tr}\left[(U_t - \tilde{U})^\dagger (U_t - \tilde{U})\right] \tag{2.12}$$

We can consider L to be the test loss which represents how close we have learned the true function of the data. In cases where we have direct access (as in chapter 4) to the entries of a unitary matrix, this loss function is a convenient choice.

Chapter 3

Comparing Methods to Parameterize and Learn Unitary Matrices

As stated in the introduction, many researchers have integrated unitary matrices into neural networks to avoid common challenges faced with exploding and vanishing gradients [2, 57, 77]. Neural networks are typically compared based on their performance in learning real-world data (e.g., images, translation, etc.). This chapter considers a more general but related learning problem: which parameterizations of unitary matrices are most effective at learning or approximating a target random unitary matrix? This more general approach is taken for two reasons. First, this general approach is agnostic to the choice of learning models and dataset. Second, learning random data can serve as a starting point for developing systems designed to learn specific datasets.

One central challenge faced when learning unitary matrices is in performing optimization on matrices in a fashion that preserves the unitarity of a matrix. Imposing unitarity is usually handled in two different ways: either by parameterizing the space of unitary matrices in a general way (e.g. [2]) or by re-unitarizing the weight matrices after each training step by projecting them onto the unitary subspace (e.g. [77]). In the framework of this paper, we use the prior method and efficiently parameterize a variety of different unitary building blocks to assemble flexible unitary representations. We investigate both the case where target unitary matrices are low dimensional (i.e 32×32) and high dimensional (i.e 1024×1024).

3.1 Unitary Matrix Parameterizations

3.1.1 Choices of unitary matrix parameterization

In order to build a unitary matrix representation, we construct networks with individual unitary layers, where each layer consists of a parameterized unitary matrix. This method of parameterizing a unitary matrix leverages the fact that the product of two unitary matrices again belongs to the group of unitary matrices. This property can be demonstrated with the help of two unitary matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{C}^{d\times d}$. Unitary matrices are defined by the property $\boldsymbol{U}\boldsymbol{U}^* = \boldsymbol{I}_d$, where $\boldsymbol{I}_d$ is the identity matrix. Using the following property of the conjugate transpose $(\boldsymbol{AB})^* = \boldsymbol{B}^*\boldsymbol{A}^*$, one can conveniently show that the product $\boldsymbol{AB}$ is indeed part of the unitary group.

$$(\boldsymbol{AB})(\boldsymbol{AB})^* = \boldsymbol{A}(\boldsymbol{BB}^*)\boldsymbol{A}^* = \boldsymbol{A}(\boldsymbol{I}_d)\boldsymbol{A}^* = \boldsymbol{AA}^* = \boldsymbol{I}_d \tag{3.1}$$

Another important property of unitary matrices is that all of their eigenvalues have magnitude equal to one. This property can be used to build very deep machine learning algorithms that naturally avoid vanishing and exploding gradients [38] [54]. Below, we list the various methods that we use to parameterize unitary matrices as individual layers of a network.

Diagonal Euler matrices

The Diagonal Euler layers feature d-parameters $w_i \in \mathbb{R}$ placed on the diagonal of a $d\times d$ matrix with zeros on the off-diagonal entries. The diagonal entries are formulated as complex exponentials yielding the following matrix:

$$\boldsymbol{D} = \mathrm{diag}(e^{iw_1}, e^{iw_2}, \ldots, e^{iw_d}) \tag{3.2}$$

where the operator diag$(\ldots)$ projects the elements of a vector onto the diagonal entries of the matrix $\boldsymbol{D} \in \mathbb{C}^{d\times d}$.

Householder reflections

Householder matrices $\boldsymbol{T}$ can be geometrically interpreted as reflections about a plane that contains the origin. The matrix is set up based on the components of the complex vector $\boldsymbol{w}$ with real and complex part of each component $w_j = a_j + ib_j$. Therefore, each Householder reflection features $2d$ parameters and is defined as:

$$\boldsymbol{T} = \boldsymbol{I}_d - 2\frac{\boldsymbol{w}\boldsymbol{w}^\dagger}{||\boldsymbol{w}||^2}\,. \tag{3.3}$$

Here, $\boldsymbol{w}^\dagger$ represents the conjugate or Hermitian transpose of $\boldsymbol{w}$.

Givens rotations

In general, Givens rotations are rotations in a plane spanned by two coordinates axes. The basic building blocks $\boldsymbol{q} \in \mathbb{C}^{2\times 2}$ are composed of two variables $w_1, w_2 \in \mathbb{R}$

$$\boldsymbol{q} = \begin{bmatrix} \cos(w_1)\, e^{iw_2} & -\sin(w_1)\, e^{-iw_2} \\ \sin(w_1) & \cos(w_1) \end{bmatrix}. \tag{3.4}$$

The chosen parameterization of $\boldsymbol{q}$ is originated in the experimental realization of a lossless beam splitter with a phase shifter at one end [65] and is the most general element of the 2-dimensional unitary group U(2). We concatenate $\log_2(d)$ matrices $\boldsymbol{Q}_\mathbf{i}$, each of them containing $d/2$ independent rotations $\boldsymbol{q}$. In this setup, each $\boldsymbol{Q}_i$ contains d trainable parameters. The dimension d is restricted to values that are a power of two.

$$\boldsymbol{Q} = \boldsymbol{Q}_\mathbf{1}\boldsymbol{Q}_\mathbf{2}\ldots\boldsymbol{Q}_{\mathbf{\log_2(\boldsymbol{d})}} \tag{3.5}$$

As a first approach, the individual Givens rotations $\boldsymbol{q}$ were placed at randomly drawn indices pairs. In addition to sequences of Givens rotations placed at random indices, we also consider sequences of Given rotations layers in butterfly arrangements (compare [24]). The butterfly algorithm originates from the Fast Fourier Transformation [52] which sets the coordinate pairs according to:

$$(2pk + j, \quad p(2k+1) + j) \tag{3.6}$$

$$\text{where } p = \frac{N}{2^i},\ k \in \{0, ..., 2^{i-1}\} \text{ and } j \in \{1, ..., p\}$$
$$\text{with } i = 1, ..., \log_2(d)\,.$$

For example, applying the butterfly algorithm for dimension $d = 4$ yields the following two matrix sequence:

$$\boldsymbol{Q_1} = \begin{bmatrix} q_{1,11} & q_{1,12} & 0 & 0 \\ q_{1,21} & q_{1,22} & 0 & 0 \\ 0 & 0 & q_{2,11} & q_{2,12} \\ 0 & 0 & q_{2,21} & q_{2,22} \end{bmatrix} \tag{3.7}$$

$$\boldsymbol{Q_2} = \begin{bmatrix} q_{3,11} & 0 & q_{3,12} & 0 \\ 0 & q_{4,11} & 0 & q_{4,12} \\ q_{3,21} & 0 & q_{3,22} & 0 \\ 0 & q_{4,21} & 0 & q_{4,22} \end{bmatrix}. \tag{3.8}$$

A graphical interpretation of the butterfly sequence is shown in figure 3-1.

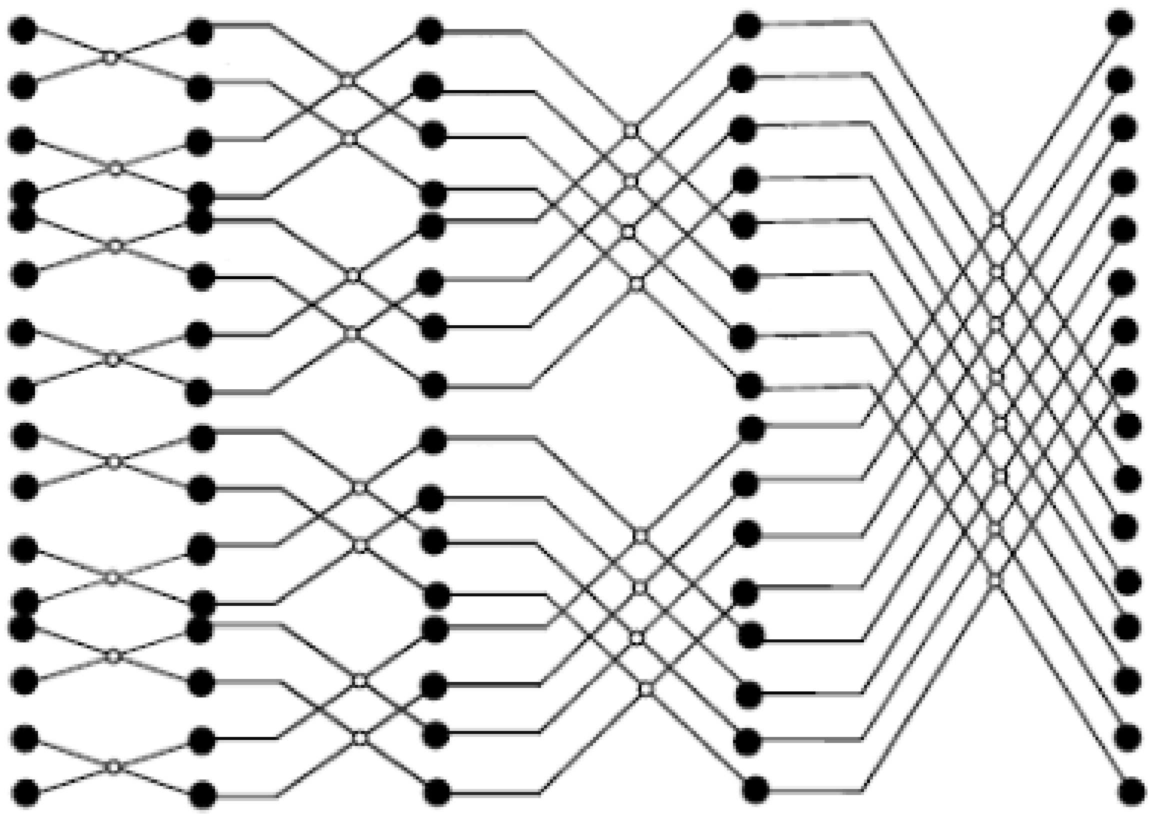


Figure 3-1: Graphical depiction of the butterfly sequence. Index pairs are indicated by white circles placed on overlapping lines.

Pauli matrices

The Pauli matrix parameterizations are based on the set of three $\mathbb{C}^{2\times 2}$ Hermitian and unitary Pauli matrices $\boldsymbol{\sigma}_{\mathbf{x}}, \boldsymbol{\sigma}_{\mathbf{y}}$ and $\boldsymbol{\sigma}_{\mathbf{z}}$.

$$\boldsymbol{\sigma}_{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \boldsymbol{\sigma}_{\mathbf{y}} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad \boldsymbol{\sigma}_{\mathbf{z}} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.9}$$

Unitary matrices can be generated by exponentiating the product of a Pauli matrix with an imaginary parameter as below:

$$e^{i\boldsymbol{\sigma} w} = \cos(w)\, \boldsymbol{I}_2 + \sin(w)\, \boldsymbol{\sigma} \tag{3.10}$$

where $\boldsymbol{I}_2$ is the identity matrix.

Note, the above simplification into trigonometric terms only holds true for matrices whose square equals the identity matrix. This is valid for all Pauli matrices since their eigenvalues are 1 and -1. Using the property above, we parameterize a $d \times d$ matrix using $d/2$ parameterized matrices as shown in formula (3.10). The resulting entries of the 2×2 matrices are then placed at a specified set of indices of the $d \times d$ matrix. For example, the 4×4 matrix below is constructed from a Pauli X placed at indices 1 and 2 and a Pauli Z placed at indices 3 and 4:

$$\boldsymbol{Q}_{\mathbf{1}} = \begin{bmatrix} e^{i\boldsymbol{\sigma}_{\mathbf{x}} w_1} & \mathbf{0} \\ \mathbf{0} & e^{i\boldsymbol{\sigma}_{\mathbf{z}} w_2} \end{bmatrix}. \tag{3.11}$$

Permutation matrix

A permutation matrix $\boldsymbol{P}$ is a $\mathbb{R}^{d\times d}$ matrix and is obtained by randomly shuffling the columns of the identity matrix $\boldsymbol{I}_d$. This matrix is unaltered during any optimization and therefore does not feature any free parameters. For instance, all possible permutation matrices for dimension $d = 3$ are the following:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{3.12}$$

DFT matrix

The discrete Fourier transform (DFT), commonly used in spectral analysis and other fields of digital signal processing, can be conveniently expressed via the unitary DFT transformation matrix $\boldsymbol{F}$. The individual entries are assigned according to $F_{jk} = \frac{1}{\sqrt{d}}(\omega^{jk})_{j,k=0..(d-1)}$. Thus, a DFT layer $\boldsymbol{F}$ as well as its inverse $\boldsymbol{F}^{-1}$ do not feature any tuneable parameters. We implement the fast Fourier transform in our simulations which performs matrix multiplication with a DFT layer in $\mathcal{O}(d\log(d))$ time.

$$\boldsymbol{F} = \frac{1}{\sqrt{d}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix} \tag{3.13}$$

Lie algebra basis

Any unitary operator can be parameterized using the Lie algebra $u(n)$ associated with its Lie group. The basis of the Lie algebra will have n^2 elements and this basis can be used to fully parameterize a unitary matrix. This method was used by Jing et al. [35] to learn unitary matrices and worked well in learning low dimensional matrices (e.g. dimension 20). However, since this method requires matrix exponentiation and custom formulas for calculating parameter derivatives (simple back-propagation is not suitable), we have chosen not to further explore this method of parameterizing unitaries in the course of this paper. Furthermore, for large matrices, a subset of the full basis must be chosen for the learning model since learning in the full parameter space is computationally infeasible. Choosing such a subset would require further model design that we leave for future work.

Conventional dense non-unitary matrix

As a point of comparison, we also include matrices parameterized by their d^2 entries. These matrices are almost never unitary. However, they are the most common matrices found in deep learning and machine learning applications (e.g. matrix corresponding to fully connected hidden layer of a neural network). We include them here to determine whether these non-unitary matrices nevertheless can approximate unitary matrices. This matrix will also serve as a benchmark for our analysis of unitary parameterized matrices.

3.1.2 Assembly of unitary networks

Instead of designing custom machine learning models to perform optimization on the parameters of unitary matrices, we leverage existing packages in deep learning and parameterize unitary matrices as customized layers of a neural network. In this setting, a parameterized unitary matrix can be considered as an individual layer of a neural network, and we can combine different parameterizations into one network by composing them as layers. In other words, each type of parameterization of a unitary matrix described in section 3.1.1 forms a single "layer" of a neural network. This is

Table 3.1: Names and descriptions of parameterized unitary networks

Name	**Description**	**Time complexity***	**Layer sequence****
EUNN	Efficient Unitary Neural Network unitary described in [38]	$\mathcal{O}(n \log n)$	1-5-6-1-4-5-6-1
Householder	Layers of d-dimensional Householder reflections	$\mathcal{O}(n)$	5
Givens-butterfly	Sequence of Givens rotations layers in butterfly arrangement from [52]	$\mathcal{O}(n)$	[3-3-3...]***
Givens-random	Layers of Givens rotation matrices; rotations placed at random indices	$\mathcal{O}(n)$	3
Givens+diagonal	Layer of random Givens rotations followed by a diagonal Euler matrix	$\mathcal{O}(n)$	3-1
Pauli-random	Layers of random Pauli X, Y, and Z matrices; parameters at random indices	$\mathcal{O}(n)$	2
dense-non-unitary	Non-unitary $d \times d$ matrix with all d^2 entries as parameters (for comparison)	$\mathcal{O}(n)$	n/a

* time complexity as a function of the number of parameters n

** refer to section 3.1.1 for descriptions of layer numbers; layer sequences may be repeated to have more parameters for learning

*** sequence is of length $\log_2(d)$ for a d dimensional target unitary

analogous to traditional neural networks where one can decompose a network into its various layers *e.g.,* convolutional or fully connected layers. For example, in our setting, a network can be composed in a hybrid fashion consisting of layers of Givens rotations and Householder reflections. The different networks that we consider are listed in table 3.1.

As in conventional neural networks, this framework provides us with flexibility in terms of combining different parameterizations as individual layers and scaling the number of parameters in our learning model by adding or removing layers from the neural network. Furthermore, one can take advantage of the various functions (including loss functions), linear algebra toolboxes, and optimization methods commonly implemented for neural networks and available in deep learning packages. In our case, optimization is efficiently performed using methods of automatic differentiation available in the python package Tensorflow [1]. The specific loss function used in this chapter is detailed in section 3.1.4.

3.1.3 Target unitary matrix and input vectors

To optimize the parameters of our unitary network, we do not directly compare the target unitary matrix with the output matrix of the layered network structure. Instead, we construct a training set of random input vectors $\boldsymbol{x}$ and target output vectors $\boldsymbol{y}$ calculated from a target unitary matrix $\boldsymbol{Y}$.

$$\boldsymbol{y} = \boldsymbol{Y}\boldsymbol{x} \tag{3.14}$$

Here, $\boldsymbol{x}$ are normalized vectors randomly sampled from a normal distribution with mean zero and standard deviation one. Within the scope of this paper, the size of our training and test sets are 8,000 and 2,000, respectively.

The random target unitary is generated according to the following scheme. We create a random matrix $\boldsymbol{N} \in \mathbb{C}^{d\times d}$. The diagonal elements are sampled from a normal distribution with a standard deviation of one. Whereas, the remaining components are drawn from a distribution with a standard deviation of one half. The random target unitary matrix $\boldsymbol{Y}$ is determined by:

$$\boldsymbol{Y} = e^{i\,n\left(\mathrm{L}(\boldsymbol{N})+\mathrm{U}(\boldsymbol{N}^{\dagger})\right)} \tag{3.15}$$

with n being a random number drawn from a uniform distribution in the interval from $-\pi$ to π. $\mathrm{L}(\boldsymbol{X})$ represents the lower triangular matrix of $\boldsymbol{N}$ and $\mathrm{U}(\boldsymbol{X}^{\dagger})$ the upper triangular matrix of the Hermitian transpose of $\boldsymbol{N}$.

3.1.4 Optimization framework

The initial parameter values for all unitary layers are generated following a standard normal distribution with mean zero and a standard deviation of one. For the dense-non-unitary layer, we initialize all matrix entries so that the variance of a column of the matrix sums to one. Thus, all parameters are sampled from a normal distribution with zero mean and variance $1/d$ where d is the dimension of the matrix.

During the training procedure, we rely on mini-batches only feeding a set number n_{batch} of randomly drawn examples of input vectors $\boldsymbol{x}$ into the back-propagation algorithm. We define a training step as one sequence of back-propagation and parameter adjustment based on one mini-batch.

The loss function calculates the mean squared error (MSE) of the difference between the network's predictions $\hat{\boldsymbol{y}}$ and the target vectors $\boldsymbol{y}$ for the current batch of size n_{batch}.

$$\mathcal{L} = \frac{1}{n_{\text{batch}}} \sum_{1}^{n_{\text{batch}}} (\boldsymbol{y} - \hat{\boldsymbol{y}})^2 \tag{3.16}$$

The optimizations are performed using the stochastic gradient descent (SGD), RMSprop (RMS), and Adam optimizers described below.

Stochastic gradient descent (SGD)

The stochastic gradient descent algorithm [68] is very closely related to the standard gradient descent algorithm. The updates of the parameters w follow the scheme:

$$w_{i,t+1} = w_{i,t} - \eta \frac{\partial \mathcal{L}_i}{\partial w_{i,t}} \tag{3.17}$$

where η represents the learning rate ($\eta = 0.3$ for our experiments) and $\mathcal{L}_i$ denotes the current loss of example i. In contrast to gradient descent, SGD performs the update according to equation (3.17) for each example i separately.

RMSprop optimizer (RMS)

RMSprop [69] integrates the idea of an adaptive learning rate. Therefore, RMSprop uses a different learning rate for each parameter following the basic notion: the smaller the gradients, the smaller the updates and vise versa.

$$\begin{gathered} \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\boldsymbol{v}_t} + \varepsilon} \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \\ \text{where } \boldsymbol{v}_t = \beta \boldsymbol{v}_{t-1} + (1 - \beta) \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right)^2 \end{gathered} \tag{3.18}$$

$\boldsymbol{v}_t$ can be interpreted as a moving average of the squared gradient featuring an exponentially decaying contribution of all previous squared gradients. Hyperparameters $\eta = 0.3$ define the learning rate, $\beta = 0.9$ defines the window size of the moving averages, and $\varepsilon = 1^{-10}$ is a small number introduced to avoid division by zero.

Adam optimizer (Adam)

Adam [44] combines RMSprop with a momentum-based gradient descent. Whereas RMSprop, described in the prior paragraph uses previous gradients to adjust the learning rate, momentum-based methods use previous gradients to calculate the current gradient. Adam combines both approaches in the following calculation procedure:

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{v}_{t-1} + (1 - \beta_1) \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \tag{3.19}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \right)^2 \tag{3.20}$$

with the help of hyperparameters $\beta_1 = 0.9$ for the momentum-like-term and $\beta_2 = 0.999$ for the RMSprop-like-term. Additionally, Adam uses bias corrected values of gradients in order to improve the calculated moving averages especially during the initial training phase.

$$\widehat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - {\beta_1}^t} \qquad \widehat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - {\beta_2}^t} \tag{3.21}$$

Summarizing the calculation scheme yields the update rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\widehat{\boldsymbol{v}}_t} + \varepsilon} \widehat{\boldsymbol{m}}_t \tag{3.22}$$

where we use the learning rate $\eta = 0.001$ for our experiments and set $\varepsilon = 1^{-8}$ to avoid division by zero.

Table 3.2: Test loss averaged across five simulations for over-parameterized networks.

	Optimizer		
Network	**Adam**	**RMS**	**SGD**
EUNN	0.705	0.917	1.948
Householder	0.009	0.013	0.012
Givens-random	< 0.001	0.002	0.130
Givens+diagonal	0.002	0.023	0.138
Pauli-random	0.009	0.008	0.008
dense-non-unitary	< 0.001	< 0.001	< 0.001

3.2 Experiments

3.2.1 Over-parameterized learning of low dimensional unitary matrices

The number of parameters of a unitary matrix grows with the dimension d as d^2. In this section, we determine whether the networks we have created can learn arbitrary matrices when the networks are over-parameterized with respect to the target unitary matrices. We consider 32×32 ($d = 32$) unitary matrices as our target, and we over-parameterize all networks by including enough layers to have at least 2048 ($2d^2$) learnable parameters in the network, i.e. each network has at least two times the number of parameters of the target unitary matrix. The Givens-butterfly network is not considered here since the butterfly arrangement produces fewer parameters than needed. For all networks, the layer sequence is repeated until the number of parameters exceeds 2048.

Table 3.2 shows the average test loss for five simulations of each over-parameterized network considered in this section. All networks except for the EUNN are able to learn the target unitary with the Adam and RMS optimizers achieving average losses of near or below 0.01 mean square error in the test set. Furthermore, figure 3-2 plots the training loss as a function of the number of training steps. The figure also shows that the Pauli-random network is the fastest to learn the target unitary. Though, networks with Givens rotations appear to outperform the Pauli-random network

Figure 3-2: Progression of training loss (log axis) as a function of the number of training steps. The Pauli-random network showed fast learning to the target unitary. Networks with Givens rotations and Householder reflections were also able to learn the target unitary though the time to convergence was slower. Five simulations were run for each network and optimizer. Note, figure should be viewed in color.

over long learning times.

From figure 3-2, it is also apparent that the Adam and the root mean square propagation (RMS) optimizers outperform optimization using the conventional stochastic gradient descent (SGD) at 10,000 training steps.

3.2.2 Learning of high dimensional unitary matrices with $\mathcal{O}(d\log_2(d))$ parameters

For large unitary matrices, performing learning on the full parameter space (i.e. d^2 parameters for a d dimensional unitary matrix) can become computationally infeasible. Thus, in this section, we consider the separate challenge of learning arbitrary matrices using networks with $\mathcal{O}(d\log_2(d))$ parameters ($\mathcal{O}(d)$ parameters per layer with $\mathcal{O}(\log_2(d))$ layers). This framework is more applicable for use in

Table 3.3: Test loss averaged across five simulations for networks with $\mathcal{O}(\log_2(d))$ layers.

	Optimizer		
Network	**Adam**	**RMS**	**SGD**
EUNN	1.953	1.955	1.994
Householder	1.992	1.989	2.039
Givens-butterfly	1.100	1.098	1.297
Givens-random	1.533	1.535	1.898
Givens+diagonal	1.890	1.883	1.958
Pauli-random	1.896	1.893	1.936
dense-non-unitary	0.930	0.877	0.877

machine learning settings where learning with fewer parameters and shorter time complexity is desired.

For this higher dimensional setting, we consider 1024×1024 ($d = 1024$) unitary matrices as our target. Layer sequences listed in table 3.1 are repeated until there are $\mathcal{O}(d \log_2(d))$ parameters. For all networks except the EUNN setup, this corresponds to repeating the layer sequence ten times. The EUNN setup has $7d$ parameters, so this layer sequence is not repeated. Furthermore, the dense-non-unitary network consists of a single matrix with d^2 parameters, and this is not repeated.

Given the larger dimension of the target unitary matrix, the learning problem is much more difficult than the prior over-parameterized setup. From table 3.3, it is clear that the Givens-butterfly network outperforms other networks in learning the target unitary achieving a mean squared error loss of around 1.1 on the test set. None of the networks are able to outperform the dense-non-unitary network. However, during training, the dense-non-unitary networks set their parameters (matrix entries) to very small values. The resulting networks are able to reduce the loss to values near 1.0 (the mean squared length of the target vectors) by constructing output vectors with small norms. Thus, the dense-non-unitary network learns by essentially eliminating information and generating a non-unitary matrix that does not preserve vector norms.

Figure 3-3: Progression of training loss as a function of the number of training steps. Givens rotations arranged in the butterfly pattern performed best in this setting. Five simulations were run for each network and optimizer. Note, figure should be viewed in color.

Figure 3-3 shows that the Givens-butterfly network not only converges to the lowest loss value, but also converges to a final solution the fastest. This holds true regardless of the choice of optimizer. From figure 3-3, it is also clear that the only other network that performs relatively well is the Givens-random network. As in the case with the over-parameterized networks in the prior section, the Adam and RMS optimizers converge to a solution in many fewer steps than conventional SGD.

3.3 Discussion

In the over-parameterized case, it appears that all networks considered except for the EUNN network are able to approximate the target unitary to low errors in the loss function. Even though all networks have roughly the same number of parameters, Pauli-random networks converge to the solution at a rate much faster than the other unitary networks, achieving convergence often even faster than dense-non-unitary networks. These results suggest that the landscapes of the loss function for the Pauli-random networks are much steeper than those of other networks. This is contrasted with results from the Givens-random and Givens+diagonal networks, where it appears that the landscapes of the loss function are shallow and steep at different points in the learning curve. It is an open question as to why the landscapes of the loss functions are different for the various networks.

In the under-parameterized case where networks are designed to have $\mathcal{O}(\log_2(d))$ layers, the Givens-butterfly network performs the best of all the unitary networks. One explanation for this result comes from random matrix theory, where as stated in section 3.1.1, $d \log_2(d)/2$ successive Givens rotations are required in order to obtain a distribution of random matrices dense in the space of rotation matrices [52]. A second explanation comes from the fact that the butterfly arrangement ensures that an interaction exists between all pairwise indices of the matrix. These two facts intuitively give a possible explanation for the strong performance of the Givens-butterfly network. However, the extent to which it outperforms other networks is partly unexpected (especially the Givens-random networks).

Of the three optimizers used, the Adam and RMS optimizers converge to critical points in the loss function at rates much faster than conventional SGD. Though the Adam and RMS optimizers sometimes also converge to lower values of the loss function, we conjecture that SGD would converge to equally good solutions if simulations are drawn out for many more training steps. More work is needed to validate this conjecture. Nevertheless, it is apparent that the landscapes of the loss function can be very shallow, and adding momentum terms as in Adam or having a changing learning rate as in RMS can significantly speed up the speed to convergence. Finally, in the case of the Adam optimizer, it appears that as a network converges to a final solution, smaller steps are needed to avoid oscillations or increases in the loss function. In future work, we hope to perform more hyperparameter tuning for the optimizers and incorporate methods to reduce learning rates over time to address these oscillations and issues with convergence.

3.4 Conclusion

Our results show that learning an arbitrary unitary matrix depends on the number of learnable parameters included in the learning model and the method used for parameterizing a unitary matrix. For the over-parameterized case, where the number of parameters in the learning model exceeds the number of parameters in the target unitary, various different networks constructed converge to a solution with very low error in both the training and test sets. Furthermore, convergence to the final solution is achieved quickly using a network parameterized by a series of random Pauli matrices. For the under-parameterized case, our results show that Givens rotations arranged in a butterfly pattern with $d \log_2(d)$ parameters can produce reasonably good convergence to a solution, though loss values in the test and training sets are still significantly larger than zero. In all cases, we find that the Adam and RMS optimizers work better than conventional SGD. Throughout our work, we observe that the landscape of the loss function is non-convex. Given this non-convexity, it is an open question whether second order optimization methods (e.g. Quasi-Newton methods) or optimization methods that do not calculate gradients (e.g. genetic algorithms) can better explore the non-convex landscapes.

For our analysis, we considered a general case where the target unitary matrices were randomly chosen. However, actual data is not random and often contains simple patterns that can potentially be easily learned. Future analysis can focus on which networks are best suited to learning simpler unitary matrices that are drawn from real-world examples. For example, learning can be performed on target unitary matrices that are composed of a small number of rotations or reflections. Alternatively, unitary matrices here can be used to learn covariance matrices generated from actual data.

Chapter 4

Learning Unitary Matrices with Alternating Hamiltonians

In this section, we shift to the quantum setting and study the hardness of learning unitary transformations by performing gradient descent on the time parameters of sequences of alternating operators. Such sequences are the basis for the quantum approximate optimization algorithm and represent one of the simplest possible settings for investigating problems of controllability.

A fundamental task in both quantum computation and quantum control is to determine the minimum amount of resources required to implement a desired unitary transformation. In this paper, we present a simple model that allows us to analyze key aspects of implementing unitaries in the context of both quantum circuits and quantum control. In particular, we implement unitaries using sequences of alternating operators of the form $e^{-iAt_K}e^{-iB\tau_K}\dots e^{-iAt_1}e^{-iB\tau_1}$. Each unitary is parameterized by the times $\{t_1, \tau_1, \dots, t_K, \tau_K\}$. This approach of parameterizing unitaries is the basis for the quantum approximate optimization algorithm (QAOA) [20, 21]. The acronym QAOA is also used to refer to the phrase "Quantum Alternating Operator Ansatz." Recently, it has been shown that quantum alternating operator unitaries can perform universal quantum computation [49]. In the infinitesimal time setting, QAOA also encompasses the more general problem of the application of time varying quantum controls [7, 9, 41, 55, 61–63, 66, 67, 70]. In this work, we study the quantum alternating operator formalism as a general framework of performing arbitrary unitary transformations.

We investigate the difficulty of learning Haar random unitaries in $U(d)$ using parameterized alternating operator sequences. Here, we find that unsurprisingly, when the number of parameters in the sequence is less than d^2, gradient descent fails to learn the random unitary. Initially, we had expected that because of the highly non-convex nature of the loss landscape, when the number of parameters in the sequence was greater than or equal to d^2, gradient descent would sometimes fail to learn the target unitary. However, our numerical experiments reveal the opposite. When the number of parameters is d^2 or greater, gradient descent *always* finds the target unitary. Moreover, we provide evidence for a "computational phase transition" at the critical point between the under-parameterized and over-parameterized cases where the number of parameters in the sequence equals d^2.

Learning Setting. Suppose we have knowledge of the entries of a unitary $\mathcal{U} \in U(d)$ and access to the Hamiltonians $\pm A$ and $\pm B$. Recent work has provided a constructive approach to build a learning sequence $\mathcal{V}(\vec{t}, \vec{\tau}) = e^{-iAt_K} e^{-iB\tau_K} \dots e^{-iAt_1} e^{-iB\tau_1}$ that can perform any target unitary $\mathcal{U}$ where $K = O(d^2)$ [50]. In this work, we ask whether optimal learning sequences for performing the target unitary $\mathcal{U}$ can be obtained by using gradient descent optimization on the parameters $\vec{t}, \vec{\tau}$ of $\mathcal{V}(\vec{t}, \vec{\tau})$. The matrices A, B are sampled from the Gaussian Unitary Ensemble (GUE) so that the algebra generated by A, B via commutation is with probability one complete in $u(d)$, *i.e.*, the system is controllable [7, 9, 41, 55, 61–63, 66, 67, 70]. The parameters $\vec{t}, \vec{\tau}$ represent the times for which the generators of $\mathcal{V}(\vec{t}, \vec{\tau})$ are applied. We assume we can apply $\pm A, \pm B$; equivalently, we can take t_j, τ_j to be positive or negative. Note that this problem formulation lies in the domain of quantum optimization algorithms such as the Quantum Approximate Optimization Algorithm [22, 25, 37, 80], the Variational Quantum Eigensolver [53, 59], and the Variational Quantum Unsampling [8] in which one varies the classical parameters in a quantum circuit to minimize some objective function.

In general, the control landscape for learning the unitary $\mathcal{U}$ is highly non-convex

[7, 9, 41, 55, 61–63, 66, 67, 70]. Gradient descent algorithms do not necessarily converge to a globally optimal solution in the parameters of a non-convex space [78], and they frequently converge instead to some undesired critical point of the loss function landscape. We study how hard it is to learn an arbitrary unitary with the quantum alternating operator formalism via gradient descent. We quantify the hardness of learning a unitary with the minimum number of parameters required in the sequence $\mathcal{V}(\vec{t},\vec{\tau})$ to perform the unitary $\mathcal{U}$. Since $\mathcal{U}$ has d^2 independent parameters, in general, at least d^2 parameters in the sequence $\mathcal{V}(\vec{t},\vec{\tau})$ are required to learn a unitary $\mathcal{U} \in U(d)$ within a desired error. Nevertheless, the non-convex loss landscape suggests that it might not be possible to learn an arbitrary $\mathcal{U}$ with gradient descent using $O(d^2)$ parameters. Our work numerically shows that exactly d^2 parameters in the sequence $\mathcal{V}(\vec{t},\vec{\tau})$ suffice to learn an arbitrary unitary $\mathcal{U}$ to a desired accuracy.

We also consider the case of learning "shallow" target unitaries of the form $\mathcal{U}(\vec{t},\vec{\tau}) = e^{-iAt_N}e^{-iB\tau_N}\ldots e^{-iAt_1}e^{-iB\tau_1}$ where the number of parameters in the target unitary is $2N \ll d^2$. For example, the simplest such target unitary is a depth-1 sequence $\mathcal{U}(t,\tau) = e^{-iAt}e^{-iB\tau}$. Such unitaries are, by definition, attainable via a shallow depth alternating operator sequence, and we look to see if it is possible to use gradient descent to obtain a learning sequence $\mathcal{V}(\vec{t},\vec{\tau})$ of the same depth that approximates the target unitary $\mathcal{U}(\vec{t},\vec{\tau})$. That is, we look at the alternating operator version of whether it is possible to learn the unitaries generated by shallow quantum circuits. We find that gradient descent typically requires d^2 parameters in the sequence $\mathcal{V}(\vec{t},\vec{\tau})$ to learn even a depth-1 unitary. This result suggests that gradient descent is not an efficient method to learn low depth unitaries.

Rabitz *et al.* consider the case of controllable quantum systems with time varying controls, including systems with drift, and show that when the controls are unconstrained (space of controls is essentially infinite dimensional), there are no sub-optimal local minima [62, 63, 70]. When the sequence of controls is finite dimensional, prior studies sometimes find traps in the control landscape [55, 66, 67]. Here, we look at the simplest possible case where the system does not have drift and the space of controls is finite dimensional. Our numerical results show that even in spaces where

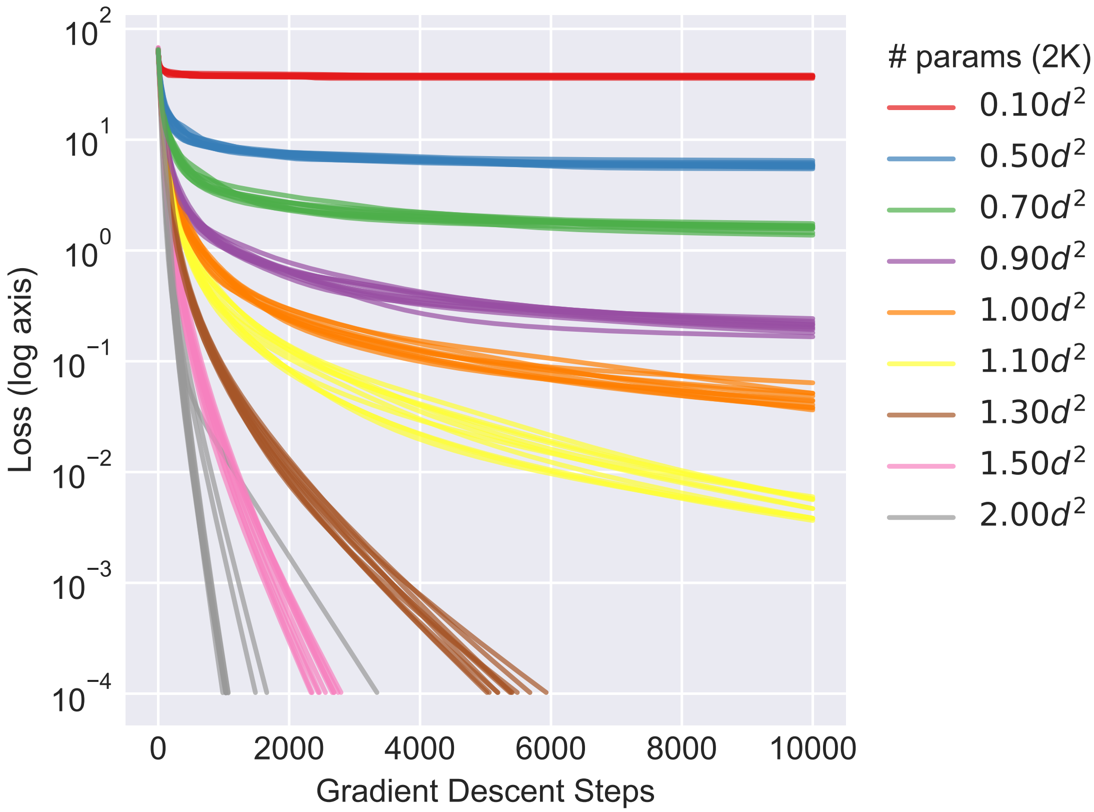
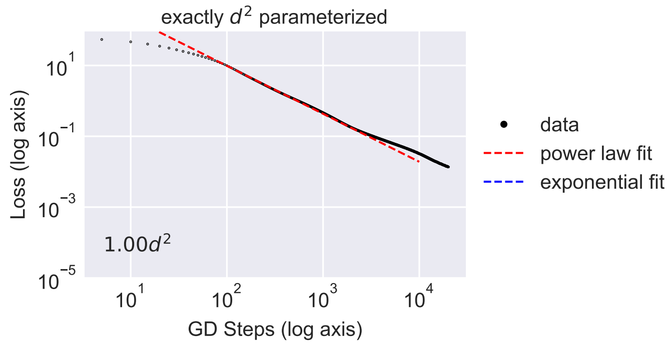


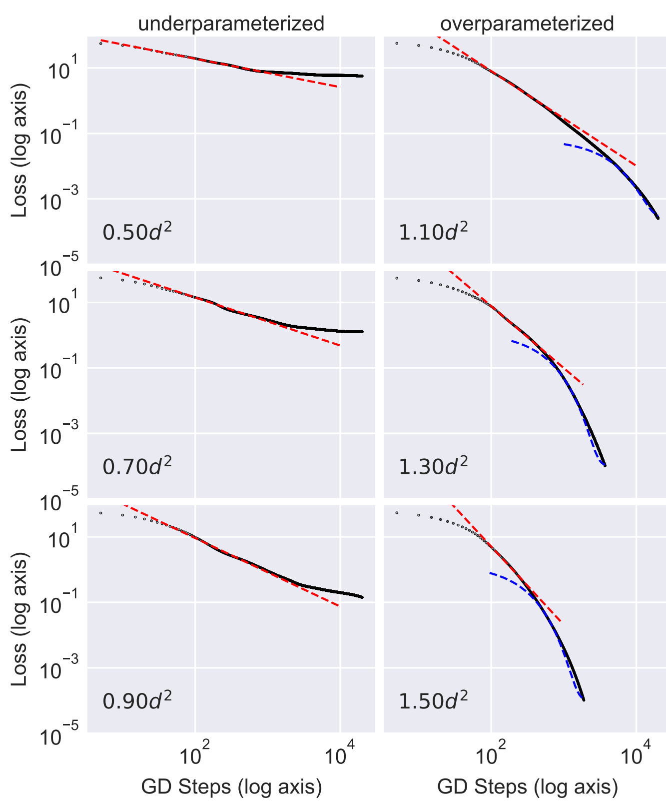
Figure 4-1: Gradient descent experiments for a Haar random target unitary $\mathcal{U}$ of dimension 32. The logarithm of the loss function $L(\vec{t}, \vec{\tau})$ with increasing gradient descent steps for learning sequences $\mathcal{V}(\vec{t}, \vec{\tau})$ with $2K$ parameters.

the dimension of the system is the minimum it can be to attain the desired unitary and the control landscape is highly non-convex, it still contains no sub-optimal local minima and gradient descent obtains the global optimal solution.

We now provide a detailed numerical analysis of the learnability of both arbitrary and shallow depth unitaries using gradient descent optimization.



(a)



(b)

Figure 4-2: Gradient descent experiments for a Haar random target unitary $\mathcal{U}$ of dimension 32 exhibit a power law convergence in the first 1,000 gradient descent steps (best fit line shown in dashed red in the plot). In the under-parameterized case, at a certain point, the gradient descent plateaus at a sub-optimal local minimum. In the over-parameterized case, after the power law regime, the gradient descent enters an exponential regime consistent with a quadratic form for the loss function in the vicinity of the global minimum (best fit line shown in dashed blue in the plot). In the critical case, $2K = d^2$, the power law persists throughout the gradient descent providing further evidence for a computational phase transition.

4.1 Numerical experiments for learning an arbitrary unitary

In this section, we present numerical experiments that aim to learn an arbitrary unitary $\mathcal{U}$ by constructing a sequence $\mathcal{V}(\vec{t},\vec{\tau}) = e^{-iAt_K}e^{-iB\tau_K}\dots e^{-iAt_1}e^{-iB\tau_1}$ and performing gradient descent on all $2K$ parameters to minimize the loss function $L(\vec{t},\vec{\tau}) = \left\|\mathcal{U} - \mathcal{V}(\vec{t},\vec{\tau})\right\|^2$. Here $\|\cdot\|$ denotes the Frobenius norm. Given access to the entries of a Haar random target unitary $\mathcal{U}$, we fix the number of parameters $2K$ and ask how many gradient descent steps S are required to construct the sequence $\mathcal{V}(\vec{t},\vec{\tau}) = e^{-iAt_K}e^{-iB\tau_K}\dots e^{-iAt_1}e^{-iB\tau_1}$ that can learn the target unitary $\mathcal{U}$ to a given accuracy or loss.

We present numerical evidence that with at least d^2 parameters in the sequence $\mathcal{V}(\vec{t},\vec{\tau})$, we can learn any selected Haar random unitary $\mathcal{U}$. Because of the highly non-convex nature of the loss landscape over the control parameters, we did not expect this result. The details of the numerical analysis are provided below.

We ran experiments for a Haar random target unitary $\mathcal{U}$ of dimension 32 while varying the $2K$ parameters in $\mathcal{V}(\vec{t},\vec{\tau})$. At each step, we compute the gradient $\nabla_{\vec{t},\vec{\tau}}L$ and perform gradient descent with fixed step size.

In Fig.(4-1), we plot the loss $L(\vec{t},\vec{\tau})$ as a function of the number of gradient descent steps S for learning sequences $\mathcal{V}(\vec{t},\vec{\tau})$ of varying depth K. When the sequence $\mathcal{V}(\vec{t},\vec{\tau})$ is under-parameterized with $2K < d^2$ parameters, we find that the loss function $L(\vec{t},\vec{\tau})$ initially decreases but then plateaus. Thus, in the under-parameterized loss landscape, we find that as expected, with high probability, the gradient descent algorithm reaches a sub-optimal value of the loss which cannot be decreased by further increasing the number of gradient descent steps.

When the number of parameters $2K$ in $\mathcal{V}(\vec{t},\vec{\tau})$ is equal to d^2 or more, we find that gradient descent always converges to the target unitary – there are apparently no sub-optimal local minima in the loss landscape. As noted above, this result was unexpected given the non-convex nature of the loss landscape. We also find that the rate of convergence grows with the degree of over-parameterization as shown

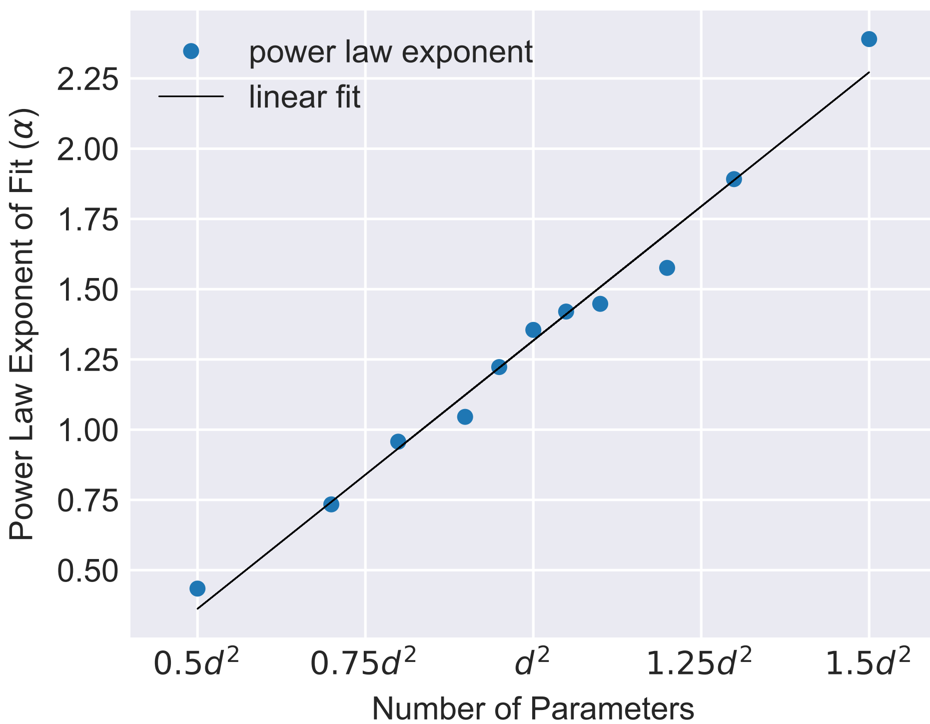


Figure 4-3: The power law rate of gradient descent α for the initial 1000 gradient descent steps grows linearly with the number of parameters ($2K$). The first 50 steps have been excluded in the fit. The slope of the best fit line is 1.9. The computational phase transition takes place at a value of $\alpha \approx 1.25$.

in Fig.(4-1). At the critical point where the number of parameters $2K = d^2$, we note the existence of a "computational phase transition." At this critical point, the learning process converges to the desired target unitary, but the rate of convergence becomes very slow. For each parameter manifold of dimension $0.1d^2 \leq 2K \leq 2d^2$, we performed ten experiments and each of the experiments has been plotted in Fig.(4-1).

In Fig.(4-2), we fit the loss $L(\vec{t}, \vec{\tau})$ over the first 1000 gradient descent steps (the first 50 steps are excluded) to a power law

$$L = C_0 S^{-\alpha} + C_1, \tag{4.1}$$

where C_0 and C_1 are constants, $L = L(\vec{t}, \vec{\tau})$ and S is the number of gradient descent steps. As shown in Fig.(4-2), the data for the initial 1000 gradient descent steps fits closely to such a power law. However, with the exception of the critical learning sequence with $2K = d^2$ parameters, the performance of gradient descent deviates from a power law fit at later steps. For the under-parameterized case, the gradient descent plateaus at a sub-optimal value of the loss. For the over-parameterized case, the power law transitions to an exponential as the gradient descent approaches the global minimum, which is consistent with the expected quadratic form of the loss function in the vicinity of the global minimum. Fig.(4-2) shows the exponential fit for the later stages of gradient descent in the over-parameterized setting. The exponential fit takes the form

$$L = C_0 e^{-r(S-S_0)} + C_1, \tag{4.2}$$

where C_0, C_1, r, and S_0 are constants (optimized during the fit), $L = L(\vec{t}, \vec{\tau})$ and S is the number of gradient descent steps.

The critical case of the sequence $\mathcal{V}(\vec{t}, \vec{\tau})$ with exactly d^2 parameters is consistent with a power law rate of convergence to the target unitary $\mathcal{U}$ during the entire gradient descent process.

The initial power law form of the gradient descent is consistent with a loss landscape that obeys the relation $\Delta L/\Delta S \propto -S^{-(\alpha+1)}$ and $\alpha \geq 0$. For example, the case $\alpha = 1$ corresponds to a power law of the form $\Delta L/\Delta S \propto -S^{-2}$. The final exponential form of convergence corresponds to the case $\alpha \to \infty$, and to a quadratic landscape where $\Delta L/\Delta S \propto -e^{-S} \propto -L$. The fitted value of α in the initial power law regime is plotted as a function of the number of parameters in Fig.(4-3). Here, we observe a linear relationship between the power law exponent α in Eq. (4.1) and the number of parameters $2K$ in $\mathcal{V}(\vec{t}, \vec{\tau})$ – *i.e.*, the larger the degree of over-parameterization, the faster the rate of convergence, and the larger the exponent in the power law.

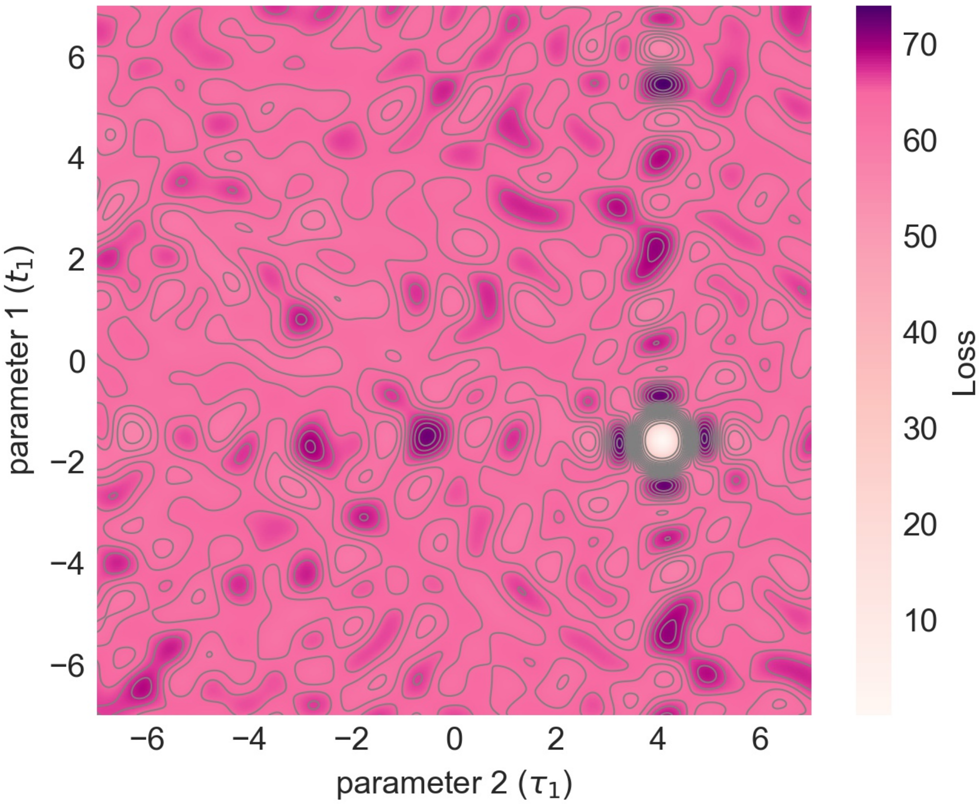


Figure 4-4: Loss function landscape when the target unitary is $e^{-iAt^*}e^{-iB\tau^*}$ where $t^* = -1.59$ and $\tau^* = 4.08$. The landscape is highly non-convex with many local minima, indicating that it is difficult to learn the target unitary with first order optimization methods such as gradient descent unless the starting point of optimization lies in the neighbourhood of the global minimum.

4.2 Learning shallow-depth unitaries

In this section, we study the learnability of low-depth alternating operator unitaries $\mathcal{U}(\vec{t},\vec{\tau}) = e^{-iAt_N}e^{-iB\tau_N}\dots e^{-iAt_1}e^{-iB\tau_1}$ where $2N \ll d^2$. Such unitaries are the alternating operator analogue of shallow depth quantum circuits. As noted above, unitaries of this form are by definition, obtainable by a learning sequence $\mathcal{V}(\vec{t},\vec{\tau})$ with depth $K \geq N$. We wish to investigate for which values of K, it is possible to learn the target unitary $\mathcal{U}(\vec{t},\vec{\tau})$ of depth N. We could reasonably hope that such a shallow depth unitary could be learned by performing gradient descent over sequences $\mathcal{V}(\vec{t},\vec{\tau})$ of depth $K = N$. We find that this is not the case. Indeed, we find that even to

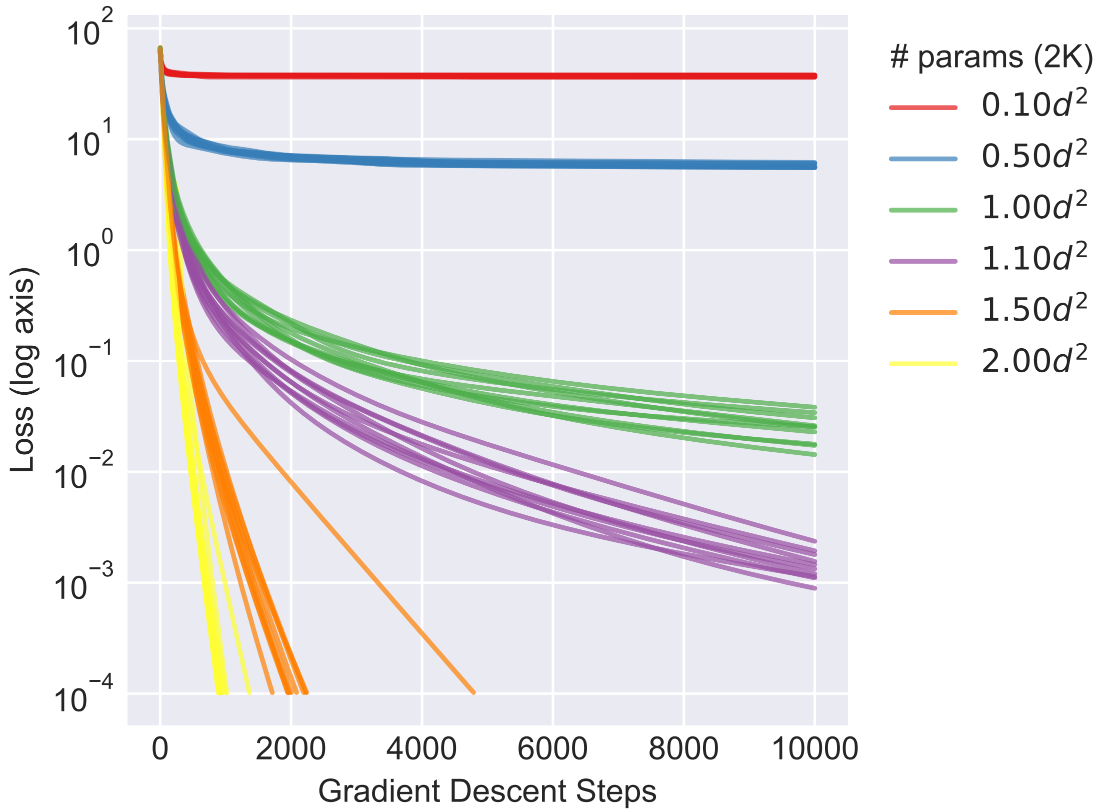


Figure 4-5: Gradient descent experiments for a low-depth unitary $\mathcal{U}(t_1^*, \tau_1^*, t_2^*, \tau_2^*)$ of dimension 32 with 4 parameters $(N{=}2)$ where $t_1^*, t_2^*, \tau_1^*, \tau_2^* \in [-2, 2]$.

learn a unitary $\mathcal{U}(\vec{t}, \vec{\tau})$ of depth $N = 1$, with high probability, we require a full depth learning sequence $\mathcal{V}(\vec{t}, \vec{\tau})$ of depth $K \geq d^2/2$ or $2K \geq d^2$ parameters in $\mathcal{V}(\vec{t}, \vec{\tau})$.

Depth $N{=}1$ unitaries take the form $\mathcal{U}(t^*, \tau^*) = e^{-iAt^*}e^{-iB\tau^*}$. In Fig.(4-4), we present the landscape of the loss function $L(t, \tau) = \left\| e^{-iAt^*}e^{-iB\tau^*} - e^{-iAt}e^{-iB\tau} \right\|^2$ which is a two dimensional parametric manifold. Here we attempt to learn the target unitary $\mathcal{U}(t^*, \tau^*)$ via a sequence $\mathcal{V}(t, \tau)$ also with two parameters. The loss function landscape is highly non-convex and contains many local sub-optimal traps. Learning the target unitary with much less than d^2 parameters using gradient descent is guaranteed only when the initial values of the parameters t, τ lie in the neighbourhood of the global minimum at $t^* = -1.59$ and $\tau^* = 4.08$. In unbounded parametric manifolds, such an optimal initialization is generally hard to achieve.

Next, we consider a target unitary $\mathcal{U}(\vec{t}^*, \vec{\tau}^*)$ with four parameters ($N = 2$). In Fig.(4-5), we find that when the sequence $\mathcal{V}(\vec{t}, \vec{\tau})$ has $2K < d^2$ parameters, the loss function plateaus with increasing gradient descent steps. This indicates that gradient descent halts at a local minimum of the loss function landscape. The rate of learning improves when $2K = d^2$ or $2K > d^2$ as in the over-parameterized domain. In this setting, the loss function rapidly converges towards the global minimum of the landscape, and the rate of convergence to the target unitary $\mathcal{U}(\vec{t}^*, \vec{\tau}^*)$ is similar to the over-parameterized case shown in Fig.(4-2).

4.3 Conclusion

We have numerically analysed the hardness of obtaining the optimal control parameters in an alternating operator sequence for learning arbitrary unitaries using gradient descent optimization. For learning a Haar random target unitary in d dimensions to a desired accuracy, we find that gradient descent requires at least d^2 parameters in an alternating operator sequence. When there are fewer than d^2 parameters in the sequence, gradient descent converges to an undesirable minimum of the loss function landscape which cannot be escaped with further gradient descent steps. This is true even for learning shallow-depth alternating operator target unitaries which are the alternating operator analogue of shallow depth quantum circuits.

Gradient descent methods generally guarantee convergence only in convex spaces. The loss function landscape for unitaries is highly non-convex, and when we began this investigation, we did not know whether gradient descent on $2K \geq d^2$ parameters in the landscape would succeed in the search for a global minimum. Indeed, we expected that gradient descent would not always converge. However, in contrast to our initial expectations, we find that when the number of parameters in the loss function landscape $2K \geq d^2$, gradient descent always converges to an optimal global minimum in the landscape. At the critical value of $2K = d^2$ parameters, we observe a "computational phase transition" characterized by a power law convergence to the global optimum.

Chapter 5

Conclusion

5.1 Applications

5.1.1 Quantum Computation and Quantum Machine Learning

Quantum computers currently perform operations that are noisy and of low fidelity [60]. In general, applying a desired set of gates may not be efficiently possible. For algorithms to realistically be implemented on near term quantum computers, different methods to perform quantum operations should be compared since some methods may be less noisy than others. The work here provides a means to construct arbitrary unitary quantum operations via parameterizations of unitary matrices that can learn the desired unitary operation. For example, in the alternating Hamiltonian setting considered in chapter 4, the control Hamiltonians A and B can be chosen as two unitary operations that have high fidelity on a quantum computer. Performing a specific unitary operation using these two control Hamiltonians may provide a method of performing arbitrary unitary operations with high fidelity.

Understanding the learnability of unitary matrices can also aid efforts to construct quantum machine learning algorithms. Speed-ups in machine learning algorithms on quantum computers can arise from embedding data into high dimensional Hilbert spaces or by implementing machine learning algorithms in variational quantum circuits [32, 51, 72]. In both of these cases, algorithms implemented in quantum computers may leverage parameterizations discussed here to form layers in a quantum learning architecture.

5.1.2 Neural Networks

As discussed in section 1.4, unitary matrices provide a straightforward method to avoid vanishing and exploding gradients in recurrent neural networks. Prior research has embedded unitary matrices into neural networks with some success [2, 38, 54, 77]. However, almost methods to embed unitary matrices into neural networks suffer can require increased computation time during training due to the parameterization of the unitary matrices. Fully parameterizing a unitary matrix often requires building networks with many layers to allow for a full parameterization. Future work can focus on methods to develop unitary optimization methods that perform optimization partially outside the unitary manifold to avoid this issue as in [52].

More broadly, unitary matrices may also prove useful in neural networks that are not recurrent. Developing very deep networks requires stability: *i.e.*, actions of many layers in composition do not cause large, chaotic changes to an input. Methods exist to preserve stability in deep networks, *e.g.*, skip connections in residual networks [33]. Unitary matrices embedded into neural networks may complement current methods or provide new structures for maintaining neural network stability since unitary matrices preserve norms.

5.2 Recommendations for Future Work

Consistent in both the quantum and classical learning settings is the finding that algorithms to learn unitary matrices are most effective when the learning algorithms are over-parameterized (*i.e.,* at least d^2 parameters are used to learn a $d \times d$ unitary matrix). Furthermore, in cases where learning algorithms are under-parameterized, we find that learning landscapes contain many traps or sub-optimal minima. Future work can focus on improving the results in the under-parameterized setting by identifying systems that can avoid sub-optimal traps. Perhaps, to achieve success in the under-parameterized setting, future research can consider problems where target unitary matrices or objective functions are limited to certain subspaces of the complete unitary manifold. Recent work on block-encoded unitary matrices and singular value transformations may be leveraged in this regard to build improved quantum machine learning algorithms [10, 26].

Furthermore, future work can broaden the scope of this thesis to consider learning unitary matrices with algorithms and cost functions not considered here. For example, all the optimization methods considered here performed some gradient-based algorithm on well defined, smooth, cost landscapes. It is a natural question whether there exist non-gradient based optimization methods which can improve upon the results here. One possible such algorithm to test would be simulated annealing and its quantum analog [39, 40].

Finally, it may be the case that real-world data is simpler to learn than the random target matrices considered here. Thus, despite the challenges faced here in learning random unitary matrices, the same algorithms implemented in more realistic settings may have better performance. Future work can and should focus on embedding unitary learning algorithms into the real-world applications discussed earlier.

Appendix A

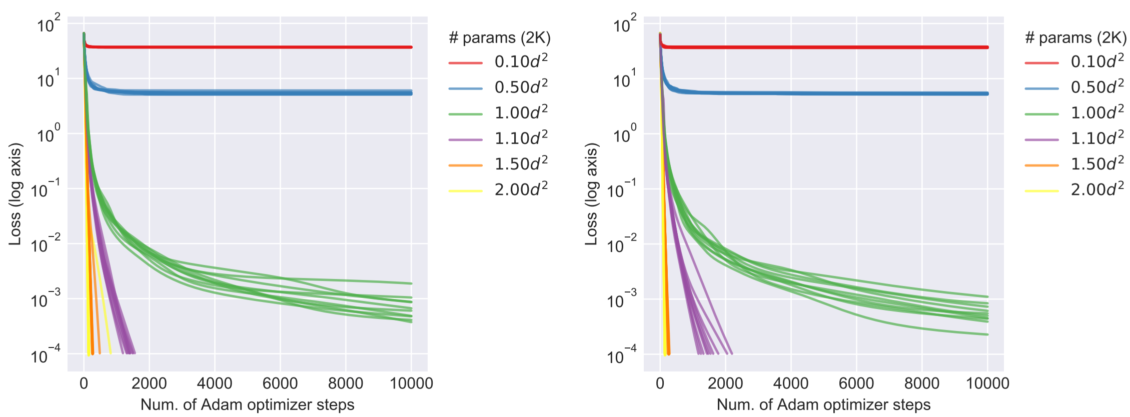
Computational Details

All experiments were performed using various different Python packages including Pytorch [58] and Tensorflow [1]. Experiments were run on a machine equipped with a Nvidia 2080 TI GPU and an Intel Core i7-9700K CPU. Calculations were performed with 64-bit floating point precision. The code used to perform the numerical experiments is available upon request.

Appendix B

Supplementary Materials to Chapter 4

B.1 Experiments using Adam optimizer



(a) (b)

Figure B-1: a) Experiments using Adam gradient descent for a Haar random target unitary $\mathcal{U}$. b) Experiments using the Adam optimizer for a low-depth target unitary $\mathcal{U}$ of dimension 32 with 8 parameters (N=4).

In addition to performing optimization using simple (vanilla) gradient descent, we performed optimization using the Adam optimizer [43], a common optimization method used in deep learning. Adaptive Moment Estimation or Adam is an upgrade of

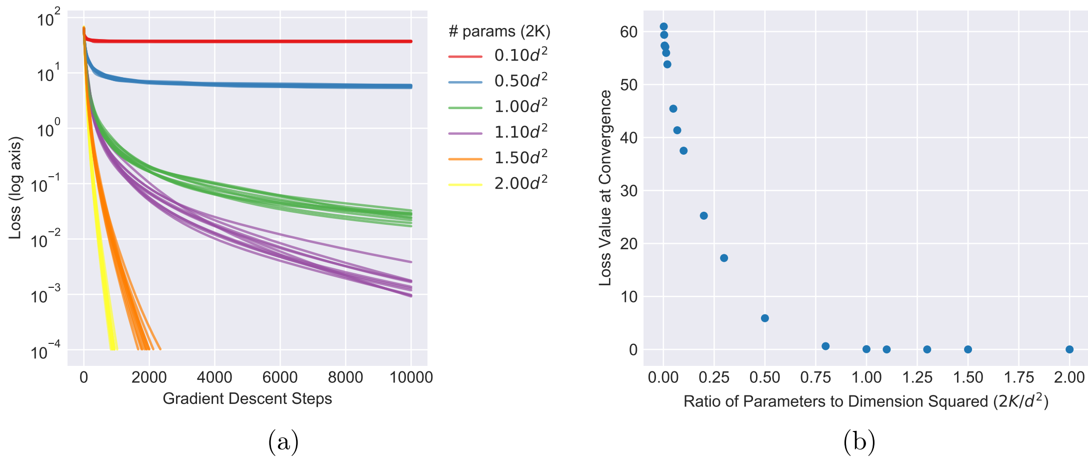


Figure B-2: a) Simple gradient descent experiments for a target unitary $\mathcal{U}$ of dimension 32 with 8 parameters (N=4). b) Value of the loss function after convergence with 10,000 steps of simple gradient descent. Experiments were performed for $\mathcal{U}$ of dimension 32 with various number of parameters.

the simple gradient descent algorithm where parameters are assigned different learning rate which are adaptively computed in every iteration of the algorithm. These updates are solely computed from first order gradients. In contrast, the learning rate is fixed for each parameter in simple gradient descent. For more on the Adam optimizer, the reader is referred to [43]. The final loss obtained for learning unitary matrices using the Adam optimizer was consistent with those obtained from simple gradient descent. However, the Adam optimizer appears to converge to a final outcome in far fewer steps. The results of our experiments are provided in Fig.(B-1). A comparison between the performance of simple and Adam gradient descent can be observed from Fig.(B-1) and Fig.(B-2).

B.2 Critical points in the under-parameterized models

When learning target unitaries using alternating operator sequences with d^2 parameters or more, gradient descent converges to a global minimum of the loss function

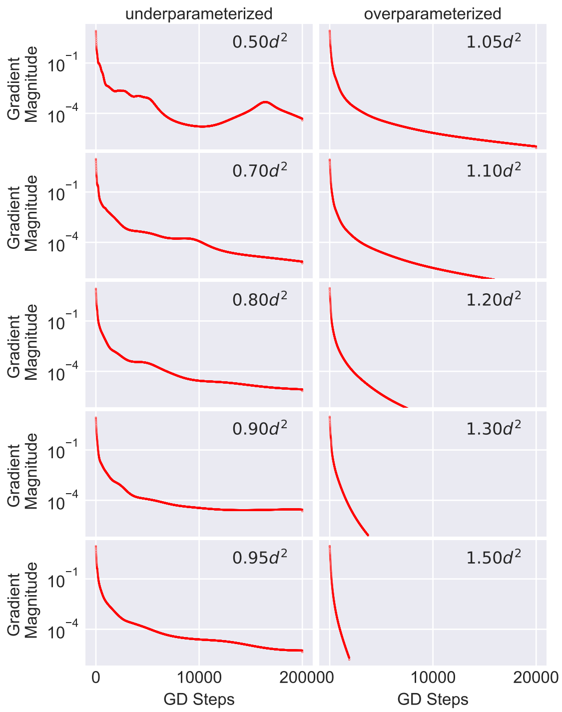


Figure B-3: Magnitude of the gradient at each step of gradient descent. In the under-parameterized setting, we find that the magnitudes can increase and decrease over the course of gradient descent. In the over-parameterized setting, we find that the magnitudes decrease, often rapidly, over the course of gradient descent. For each parameter setting, a single experiment was performed which has been plotted here.

landscape. When learning with under-parameterized models, we find that gradient descent plateaus at a non-zero loss function value. In the under-parameterized setting, we further explore how the loss function changes over the course of gradient descent by investigating the magnitude of the gradients. In the under-parameterized setting, we find that the magnitude of the gradients can both increase and decrease over the course of gradient descent, suggesting that the path of gradient descent passes in the vicinity of saddle points in the loss landscape. In the over-parameterized setting, the magnitudes of the gradients monotonically decrease with increasing gradient descent steps, suggesting that in this case, the path of gradient descent does not explore saddle points. The results of our findings are presented in Fig.(B-3).

B.3 A greedy algorithm

As noted in the text, we find that gradient descent algorithms require d^2 parameters in the sequence $\mathcal{V}(\vec{t},\vec{\tau})$ to learn a low-depth unitary $\mathcal{U}(\vec{t}^*,\vec{\tau}^*) = e^{-iAt_N^*}e^{-iB\tau_N^*}\dots e^{-iAt_1^*}e^{-iB\tau_1^*}$ where $N = O(1)$. This suggests that such low-depth unitaries are intrinsically hard to learn with less than d^2 parameters using gradient descent. We also considered a simple greedy algorithm for performing a low-depth target unitary $\mathcal{U}(\vec{t}^*,\vec{\tau}^*)$. Let $\mathcal{V}_q(\vec{t},\vec{\tau}) = e^{-iAt_q}e^{-iB\tau_q}\dots e^{-iAt_1}e^{-iB\tau_1}$. The first step of the greedy algorithm begins with $q = 1$ and uses gradient descent to optimize the parameters t_1 and τ_1. The next step of the algorithm at $q = 2$ performs gradient descent starting from the initial values $t_2 = \tau_2 = 0$ and t_1, τ_1 which are the optimal values obtained in the previous step. The greedy algorithm then continues, and at each step, q is incremented by 1. At the qth step, the initial starting points for gradient descent are $t_q = \tau_q = 0$ and the remaining parameters $\{t_i, \tau_i\}_{1\leq i\leq q-1}$ are the optimal values obtained at the end of the previous step. We present the pseudocode of the greedy algorithm below.

Algorithm 1 Greedy Algorithm

Input: Low-depth target unitary $\mathcal{U}(\vec{t^*}, \vec{\tau^*}) \in U(d)$ and GUE matrices $\pm A, \pm B$.

Initialize: Parameters $t_1, \tau_1 = 0$. $\mathcal{V}_0(\vec{t}, \vec{\tau}) = I$. Loss $= a_0 = \left\|\mathcal{U}(\vec{t^*}, \vec{\tau^*}) - I\right\|^2$.

1: **while** $q \leq d^2/2$ **do**

2: Construct the unitary $\mathcal{V}_{q-1}e^{-iAt_q}e^{-iB\tau_q}$ with parameters t_q, τ_q. Loss $= a_q = \left\|\mathcal{U}(\vec{t^*}, \vec{\tau^*}) - \mathcal{V}_{q-1}e^{-iAt_q}e^{-iB\tau_q}\right\|^2$.

3: Perform gradient descent on $\{t_i, \tau_i\}_{1\leq i\leq q}$ to minimize a_q starting from $\{t'_i, \tau'_i\}_{1\leq i\leq q-1}$ in $\mathcal{V}_{q-1}$ and $t_q = \tau_q = 0$.

4: Let $\{t'_i, \tau'_i\}_{1\leq i\leq q}$ be the optimal parameters. Updated loss $= a'_q = \left\|\mathcal{U}(\vec{t^*}, \vec{\tau^*}) - e^{-iAt'_q}e^{-iB\tau'_q}\cdots e^{-iAt'_1}e^{-iB\tau'_1}\right\|^2$.

5: **if** $a'_q \leq \varepsilon$, the sequence $\mathcal{V}_q(\vec{t'}, \vec{\tau'})$ converges to $\mathcal{U}(\vec{t^*}, \vec{\tau^*})$ and let Q = q.

6: **else** continue.

7: **end while**

Output: $\mathcal{V}_Q(\vec{t'}, \vec{\tau'})$ such that $\left\|\mathcal{U}(\vec{t^*}, \vec{\tau^*}) - \mathcal{V}_Q(\vec{t'}, \vec{\tau'})\right\|^2 \leq \varepsilon$.

We investigated the performance of the greedy algorithm for systems of up to five qubits in a restricted area of the loss function landscape. In particular, we considered the parameters $t^*_j, \tau^*_j \in [-2, 2]$ in a low-depth target unitary $\mathcal{U}(\vec{t^*}, \vec{\tau^*})$. In this setting, we find that with a probability that decreases as a function of the number of qubits, the greedy algorithm can construct a sequence $\mathcal{V}_Q(\vec{t}, \vec{\tau})$ where $Q \ll d^2/2$. In contrast, gradient descent experiments require d^2 parameters in $\mathcal{V}(\vec{t}, \vec{\tau})$ to learn $\mathcal{U}(\vec{t^*}, \vec{\tau^*})$. That is, the greedy algorithm does indeed sometimes find low-depth target unitaries even in cases where simple gradient descent on under-parameterized sequences fails. For a system of five qubits, the success probability of the greedy algorithm to learn a target unitary $\mathcal{U}(\vec{t^*}, \vec{\tau^*})$ of depth $N = 2$ (*i.e.*, with 4 parameters) using less than 50 parameters in $\mathcal{V}_Q(\vec{t}, \vec{\tau})$ is around 0.1.

Bibliography

[1] TensorFlow | Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation.

[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary Evolution Recurrent Neural Networks. *arXiv:1511.06464 [cs, stat]*, May 2016. arXiv: 1511.06464.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[4] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1:2012, 2012.

[5] Mathias Berglund, Tapani Raiko, Mikko Honkala, Leo Kärkkäinen, Akos Vetek, and Juha Karhunen. Bidirectional Recurrent Neural Networks as Generative Models - Reconstructing Gaps in Time Series. *arXiv:1504.01575 [cs]*, November 2015. arXiv: 1504.01575.

[6] Ugo Boscain, Jean-Paul Gauthier, Francesco Rossi, and Mario Sigalotti. Approximate controllability, exact controllability, and conical eigenvalue intersections for quantum mechanical systems. *Communications in Mathematical Physics*, 333(3):1225–1239, 2015.

[7] C. Brif, R. Chakrabarti, and H. Rabitz. Control of quantum phenomena: past, present and future. *New Journal of Physics*, 12(7):075008, 2010.

[8] J. Carolan, M. Mohseni, J. P. Olson, M. Prabhu, C. Chen, D. Bunandar, M. Y. Niu, N. C. Harris, F. Wong, M. Hochberg, et al. Variational quantum unsampling on a quantum photonic processor. *Nature Physics*, pages 1–6, 2020.

[9] R. Chakrabarti and H. Rabitz. Quantum control landscapes. *International Reviews in Physical Chemistry*, 26(4):671–735, 2007.

[10] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: improved regression techniques via faster Hamiltonian simulation. *arXiv:1804.01973 [quant-ph]*, September 2018. arXiv: 1804.01973.

[11] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259 [cs, stat]*, October 2014. arXiv: 1409.1259.

[12] Juan Ignacio Cirac, Peter Zoller, H Jeff Kimble, and Hideo Mabuchi. Quantum state transfer and entanglement distribution among distant nodes in a quantum network. *Physical Review Letters*, 78(16):3221, 1997.

[13] Mehmet Dağlı, Domenico D'Alessandro, and Jonathan DH Smith. A general framework for recursive decompositions of unitary quantum evolutions. *Journal of Physics A: Mathematical and Theoretical*, 41(15):155302, 2008.

[14] Domenico d'Alessandro. *Introduction to quantum control and dynamics*. CRC press, 2007.

[15] Sebastian Deffner and Steve Campbell. Quantum speed limits: from heisenberg's uncertainty principle to optimal quantum control. *Journal of Physics A: Mathematical and Theoretical*, 50(45):453001, 2017.

[16] G Dirr and U Helmke. Lie theory for quantum control. *GAMM-Mitteilungen*, 31(1):59–93, 2008.

[17] P Dita. Factorization of unitary matrices. *Journal of Physics A: Mathematical and General*, 36(11):2781, 2003.

[18] Daoyi Dong and Ian R Petersen. Quantum control theory and applications: a survey. *IET Control Theory & Applications*, 4(12):2651–2671, 2010.

[19] Shi-Hai Dong. *Factorization method in quantum mechanics*, volume 150. Springer Science & Business Media, 2007.

[20] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. *arXiv:1411.4028*, 2014.

[21] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem. *arXiv:1412.6062*, 2014.

[22] E. Farhi and A. W. Harrow. Quantum supremacy through the quantum approximate optimization algorithm. *arXiv:1602.07674*, 2016.

[23] Thomas Frerix and Joan Bruna. Approximating Orthogonal Matrices with Effective Givens Factorization. *arXiv:1905.05796 [math]*, May 2019. arXiv: 1905.05796.

[24] Alan Genz. Methods for Generating Random Orthogonal Matrices. June 1999.

[25] A. Gilyén, S. Arunachalam, and N. Wiebe. Optimizing quantum optimization algorithms via faster quantum gradient computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1425–1444. SIAM, 2019.

[26] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. *arXiv:1806.01838 [quant-ph]*, June 2018. arXiv: 1806.01838.

[27] Steffen J Glaser, Ugo Boscain, Tommaso Calarco, Christiane P Koch, Walter Köckenberger, Ronnie Kosloff, Ilya Kuprov, Burkhard Luy, Sophie Schirmer, Thomas Schulte-Herbrüggen, et al. Training schrödinger's cat: quantum optimal control. *The European Physical Journal D*, 69(12):279, 2015.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[29] Daniel Gottesman. An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation. *arXiv:0904.2557 [quant-ph]*, April 2009. arXiv: 0904.2557.

[30] David J. Griffiths. *Introduction to Quantum Mechanics*. Pearson Prentice Hall, Upper Saddle River, NJ, 2nd edition edition, April 2004.

[31] Brian Hall. *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer, Cham ; New York, 2nd ed. 2015, corr. 2nd printing 2016 edition edition, August 2016.

[32] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, March 2019.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, December 1997.

[35] Stephanie L. Hyland and Gunnar Rätsch. Learning Unitary Operators with Help From u(n). *arXiv:1607.04903 [cs, stat]*, January 2017. arXiv: 1607.04903.

[36] Cecilia Jarlskog. Recursive parametrization and invariant phases of unitary matrices. *Journal of mathematical physics*, 47(1):013507, 2006.

[37] Z. Jiang, E. Rieffel, and Z. Wang. A QAOA-inspired circuit for Grover's unstructured search using a transverse field. *arXiv:1702.0257*, 2017.

[38] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNNs. *arXiv:1612.05231 [cs, stat]*, April 2017. arXiv: 1612.05231.

[39] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, May 2011.

[40] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355–5363, November 1998.

[41] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance*, 172(2):296–305, 2005.

[42] Navin Khaneja, Steffen J Glaser, and Roger Brockett. Sub-riemannian geometry and time optimal control of three spin systems: quantum gates and coherence transfer. *Physical Review A*, 65(3):032301, 2002.

[43] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. arXiv: 1412.6980.

[45] Martin Larocca, Esteban A Calzetta, and Diego A Wisniacki. Navigating on quantum control solution subspaces. *arXiv preprint arXiv:2001.05941*, 2020.

[46] Martín Larocca, Pablo M Poggi, and Diego A Wisniacki. Quantum control landscape for a two-level system near the quantum speed limit. *Journal of Physics A: Mathematical and Theoretical*, 51(38):385305, 2018.

[47] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[48] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[49] S. Lloyd. Quantum approximate optimization is computationally universal. *arXiv:1812.11075*, 2018.

[50] S. Lloyd and R. Maity. Efficient implementation of unitary transformations. *arXiv:1901.03431*, 2019.

[51] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. *arXiv:2001.03622 [quant-ph]*, February 2020. arXiv: 2001.03622.

[52] Michael Mathieu and Yann LeCun. Fast Approximation of Rotations and Hessians matrices. *arXiv:1404.7195 [cs]*, April 2014. arXiv: 1404.7195.

[53] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[54] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections. *arXiv:1612.00188 [cs]*, June 2017. arXiv: 1612.00188.

[55] K. Moore, M. Hsieh, and H. Rabitz. On the relationship between quantum control landscape structure and optimization complexity. *The Journal of chemical physics*, 128(15):154117, 2008.

[56] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge ; New York, 10th anniversary ed. edition edition, January 2011.

[57] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the diculty of training recurrent neural networks. page 9.

[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[59] A. Peruzzo, J. McClean, P. Shadbolt, M. H. Yung, X. Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[60] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. arXiv: 1801.00862.

[61] H. Rabitz, T. S. Ho, M. Hsieh, R. Kosut, and M. Demiralp. Topology of optimally controlled quantum mechanical transition probability landscapes. *Physical Review A*, 74(1):012721, 2006.

[62] H. Rabitz, M. Hsieh, and C. Rosenthal. Landscape for optimal control of quantum-mechanical unitary transformations. *Physical Review A*, 72(5):052337, 2005.

[63] H. Rabitz, M. M. Hsieh, and C. M. Rosenthal. Quantum optimally controlled transition landscapes. *Science*, 303(5666):1998–2001, 2004.

[64] Viswanath Ramakrishna and Herschel Rabitz. Relation between quantum computing and quantum controllability. *Physical Review A*, 54(2):1715, 1996.

[65] Michael Reck, Anton Zeilinger, Herbert J. Bernstein, and Philip Bertani. Experimental realization of any discrete unitary operator. *Physical Review Letters*, 73(1):58–61, July 1994.

[66] G. Riviello, K. M. Tibbetts, C. Brif, R. Long, R. B. Wu, T. S. Ho, and H. Rabitz. Searching for quantum optimal controls under severe constraints. *Physical Review A*, 91(4):043401, 2015.

[67] G. Riviello, R. Wu, Q. Sun, and H. Rabitz. Searching for an optimal control in the presence of saddles on the quantum-mechanical observable landscape. *Physical Review A*, 95(6):063418, 2017.

[68] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951.

[69] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*, June 2017. arXiv: 1609.04747.

[70] B. Russell, H. Rabitz, and R. Wu. Quantum control landscapes are almost always trap free. *Journal of Physics A: Mathematical and Theoretical*, 50(20):205302, 2017.

[71] J. J. Sakurai and Jim J. Napolitano. *Modern Quantum Mechanics*. Pearson, Boston, 2 edition edition, July 2010.

[72] Maria Schuld, Alex Bocharov, Krysta Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *arXiv:1804.00633 [quant-ph]*, April 2018. arXiv: 1804.00633.

[73] William Shakespeare. *A Midsummer Night's Dream*. November 2014.

[74] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[75] Gilbert Strang. *Introduction to Linear Algebra, Fifth Edition*. Wellesley-Cambridge Press, Wellesley, MA, fifth edition edition edition, June 2016.

[76] Ramon Van Handel, John K Stockton, and Hideo Mabuchi. Feedback control of quantum state reduction. *IEEE Transactions on Automatic Control*, 50(6):768–780, 2005.

[77] Scott Wisdom, Thomas Powers, John R. Hershey, Jonathan Le Roux, and Les Atlas. Full-Capacity Unitary Recurrent Neural Networks. *arXiv:1611.00035 [cs, stat]*, October 2016. arXiv: 1611.00035.

[78] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar. Adaptive methods for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 9793–9803, 2018.

[79] Robert Zeier and Thomas Schulte-Herbrüggen. Symmetry principles in quantum systems theory. *Journal of mathematical physics*, 52(11):113510, 2011.

[80] L. Zhou, S. T. Wang, S. Choi, H. Pichler, and M. D. Lukin. Quantum approximate optimization algorithm: performance, mechanism, and implementation on near-term devices. *arXiv:1812.01041*, 2018.