A MACHINE LEARNING APPROACH TO IMPROVE SCALABILITY AND ROBUSTNESS OF VARIATIONAL QUANTUM CIRCUITS

by

Ankit Kulshrestha

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Winter 2024

© 2024 Ankit Kulshrestha
All Rights Reserved

A MACHINE LEARNING APPROACH TO IMPROVE SCALABILITY AND ROBUSTNESS OF VARIATIONAL QUANTUM CIRCUITS

by

Ankit Kulshrestha

Approved: _____

Weisong Shi, Ph.D.
Chair of the Department of Computer Science

Approved: _____

Levi Thompson, Ph.D.
Dean of the College of Engineering

Approved: _____

Louis F. Rossi, Ph.D.
Vice Provost for Graduate and Professional Education and
Dean of the Graduate College

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Ilya Safro, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Xi Peng, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Yuri Alexeev, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Matthew Mauriello, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

This dissertation is a result of a concentrated effort spanning two years of research in quantum computing algorithms from a machine learning practitioner's perspective. This work would not have been possible without the constant support and guidance from my advisor Dr. Ilya Safro. In some respects, I owe the very existence of the algorithms contained in this dissertation to him because he took an interest in a lost kid who was passionate about research but did not know how to conduct it some years ago. I would also like to thank my collaborators Dr. Yuri Alexeev, Marco Cerezo, Xiaoyuan Liu and Hayato Ushijima-Mwesigwa who helped me hone my research skills through insightful discussions and helpful comments in various works we published together.

No work is done without the love and support of people around you. I'm grateful to my parents and sister for supporting me in my low moments and celebrating my high ones. Last but not the least, I would like to thank my ever suffering wife Trapti, who has been patient throughout the ups and downs in this journey. She is my true friend and motivator in completing this work.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Quantum computing is an emerging new field that aims to leverage the power of a "quantum computer" to solve problems which are currently considered to NP-Hard or NP-Complete. The key idea is to encode inputs as quantum states and device a system where the measured outcomes correspond to a solution of the given problem. While a fault-tolerant quantum computer is still a theoretical possibility, we are able to evaluate the potential of quantum algorithms by running them on a class of devices called Noisy Intermediate Scale Quantum (NISQ) computers.

The advantage of having access to NISQ computers is that they allow for immediate verification of the speedup provided by a proposed quantum algorithm. However, there are significant downsides to the current generation of these devices. Most notable of them are limited gate depth, high sensitivity to noise, ability to scale to only a few number of qubits and a tendency to get stuck in "barren plateaus". In this research proposal, we introduce and define some key problems encountered in the simulation of quantum algorithms on NISQ devices. We then propose mitigating solutions inspired from machine learning methods and show the efficacy of our methods by simulating different types of variational algorithms on NISQ devices.

Chapter 1

INTRODUCTION

In this section we briefly introduce the key principles of quantum computing along with the circuit model that is currently ubiquitous in all quantum algorithms. We then introduce the dominant class of quantum algorithms dubbed as variational quantum algorithms. Finally, we introduce the research questions that we seek to address in the rest of the proposal.

1.1 Fundamental Principles of Quantum Computing

The atomic quantity in a quantum computer is the *qubit*. Informally, a qubit is a compact description of a system that consist of two outcomes *in-superposition* i.e. there is a finite probability associated with each outcome. Using the Dirac notation a general qubit $|\psi\rangle$ may be written as:

$$|\psi\rangle := \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

Where $\alpha, \beta \in \mathbb{C}$ and $|0\rangle = \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ and $|1\rangle = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$. One of the fundamental principles of quantum computing is that $|\alpha|^2 + |\beta|^2 = 1$ where $|\alpha|^2 = \alpha^\dagger\alpha$. In other words the coefficients of superposition always describe a probability distribution. Once we have defined the input as a quantum state $|\psi\rangle$, we perform some computation and then make a *measurement* on the resulting quantum state. A measurement is defined as a collection of operators $\{M_k\}$ that satisfy:

$$\sum_k M_k^\dagger M_k = I$$

And the k^{th} outcome is a scalar given by: $p(k) := \langle\phi|M_k^\dagger M_k|\phi\rangle$. The resulting state of the system after measurement is:

$$|\phi\rangle' = \frac{M_k|\phi\rangle}{\sqrt{p(k)}} \tag{1.2}$$

Making the example specific for the qubit in Equation 1.1, the probabilites can be computed as:

$$p(0) = \langle\psi|0\rangle\langle 0|\psi\rangle = |\langle 0|\psi\rangle|^2 = |\alpha|^2 \tag{1.3}$$

$$p(1) = \langle\psi|1\rangle\langle 1|\psi\rangle = |\langle 1|\psi\rangle|^2 = |\beta|^2 \tag{1.4}$$

In practice, we assume that the operators are positive operators that satisfy $\sum_k E_k = I$, where $E_k = M_k^2$. We denote these operator as **Positive Operator Value Measurements (POVM)**. A downside of Equation 1.2 is that once the qubit has been measured the resulting state does not change. This means that an input state needs to be prepared multiple times to make statistical estimations of the true outcome.

1.2 The Circuit Model of Computation

The current model of quantum computing is based on the so-called circuit model. In this model, an n-qubit system can be acted upon by an n-dimensional linear operator. This n-dimensional operator is called a quantum gate if it satisfies the *unitary property* i.e. for a linear operator $U \in \mathbb{C}^n$, it satisfies $U^\dagger U = I$.

A quantum circuit is composed of multiple single qubit gates acting locally on their input qubits and *entanglement* operations that entangle pairs of qubits. A sub-structure of single qubit gates and entanglement operators can be repeated multiple times to form a layered ansatz. A measurement is typically performed at the last layer with a POVM observable. A fundamental set of single qubit gates is the Pauli-set: $\{I, \sigma_x, \sigma_y, \sigma_z\}$ where:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.5}$$

These fundamental gates give rise to rotation gates $R_k(\theta) = e^{-i\sigma_k\theta/2}$, with $k \in \{x, y, z\}$ and θ being the angle of rotation. The control gates are multi-qubit gates that act

simultaneously on multiple qubits. One key example is the CNOT (or the controlled-NOT) gate that changes the state of the target qubit only when the control (or the source) qubit is in state $|1\rangle$. The matrix representation is given as:

$$CNOT_{0,1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.6}$$

Where the computational basis is formed over $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.

No Cloning Theorem: An important result in quantum computing restricts one of the most fundamental operations in classical computing - the copy operation. The theorem states that no unitary operator can only copy another quantum state into other if it is orthogonal or parallel to the input quantum state. Thus, arbitrarily rotated quantum states can not be copied into any other quantum state. This limitation along with the destructive measurement property of quantum computing makes it difficult to get one-shot results of a computation. A fully robust quantum computer must then be able to make as few circuit calls as possible in order to achieve a superior speedup over classical computing algorithms.

1.3 Variational Quantum Algorithms

In the current era of NISQ computing there is a dominant class of algorithms that aims to leverage hybrid computing in order to solve problems using quantum circuits. These algorithms are called Variational Quantum Algorithms (VQAs)[1]. The "variational" part of these algorithms indicates that the output changes in response to change of input parameters. They typically consist of a Variational Quantum Circuit (VQC) that runs on a quantum computer and produces an estimate of the given objective function along with its gradient. A classical computer updates the parameters using an update rule (typically adjusting the parameters in the negative direction of the gradient). This process is shown in Figure 1.1. At the end of the process, the resulting

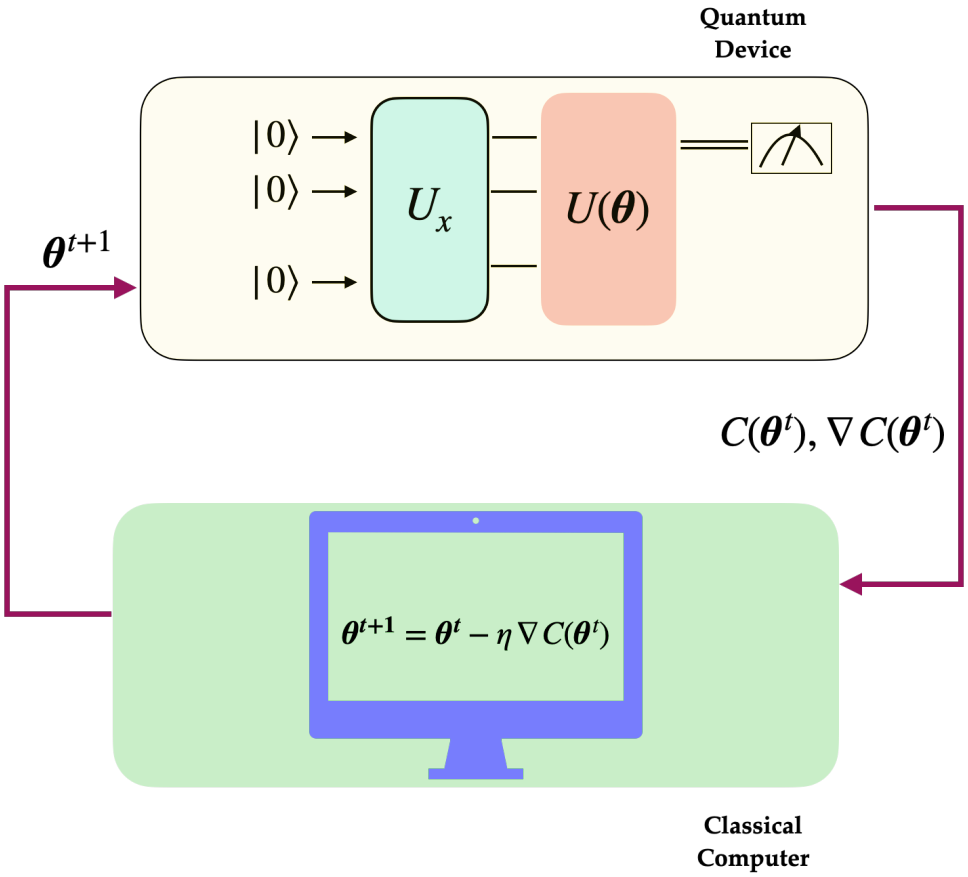


Figure 1.1. *A diagrammatic representation of a VQA running on a quantum computer with the optimizer running on a classical computer.*

parameters are supposed to be an *approximation* to the exact solution of the problem encoded by the objective function. A key difference from other quantum algorithms is that a new quantum state is produced as $|\phi\rangle = U(\boldsymbol{\theta})|\psi\rangle$, where $\boldsymbol{\theta} \in \mathbb{R}^{nL}$ with n being number of qubits and L being the number of layers. The unitary operator can be written as a product:

$$U(\boldsymbol{\theta}) = \prod_{i=1}^{nL} U(\theta_i)$$

For a given POVM $\hat{\boldsymbol{O}}$, the output can be expressed as an expectation:

$$\langle f(|\phi\rangle, \boldsymbol{\theta})\rangle = \langle\phi|\hat{\boldsymbol{O}}|\phi\rangle = \langle\psi|U(\boldsymbol{\theta})^\dagger \hat{\boldsymbol{O}} U(\boldsymbol{\theta})|\psi\rangle \tag{1.7}$$

This expectation is used as an output from the quantum circuit for evaluating the

quality of solution at the given set of parameters. The flexible architecture of VQAs make them suitable for solving optimization problems in different fields. Some of the examples include approximating solutions to such NP-hard problems as graph max-cut[2,3], minimum graph partitioning[4], community detection[5]? , estimating the ground state energy of molecules[6], portfolio optimization[7], pattern recognition[8], unsupervised data compression[9] and even circuit architecture search[10].

1.4 Research Questions

The strength of the VQAs lies in the hybrid nature of computation. However, they are still limited from achieving their true potential in solving larger instances of complex problems due to issues that are notorious in VQCs running on NISQ devices. These issues are not merely a result of the limitations of the current generation of devices, but rather they are set to worsen with the advent of better NISQ computers.

One of the key issues is the *trainability* of the quantum circuits. It has been shown empirically and analytically that VQCs suffer from a tendency to get stuck in "barren plateaus"[11–13] i.e. regions of optimization surface from where no further descent is possible and the set of parameters associated with the region produce sub-optimal output. Worse, this tendency increases as the number of qubits or layers in a circuit increases. This implies that after a certain depth, the circuit may not be able to learn anything useful about the input.

In this work, we aim to provide solutions to the issue of trainability of VQCs. Specifically, we formulate the following questions:

- *Can better parameter initialization help alleviate barren plateaus?*
- *Can variational circuits be trained without computation of gradients?*
- *Can an online-pruning procedure help enhance trainability without significantly impacting the performance of VQC?*

Chapter 2

ALLEVIATING BARREN PLATEAUS WITH BETTER INITIALIZATION

We will present a solution to the first research question posed in Chapter 1. More specifically, we will consider a solution to the barren plateau problem by choosing the right initialization.

2.1 Barren Plateau Problem

We define the barren plateau phenomenon as follows:

Definition 1. *A VQC is said to be stuck in a barren plateau if any part of a circuit exhibits a two-design due to which the variance of the gradient decays exponentially with increasing system size.*

A circuit with that acts upon a system of d qubits corresponds to sampling a unitary matrix from the unitary group $U(d)$. For any measure dU on this group, the k^{th} moments can be expressed as in[14] $M_k(dU) = \int_{U(d)} dU U_{i1,j1} \dots U_{ik,jk} U^{\dagger}_{i1',j1'} \dots U^{\dagger}_{ik',jk'}$, where ik, jk are the row and column indices of the k^{th} unitary matrix. If the measure is a left invariant measure like Haar-measure, i.e. $dU = dU_H$ then, the first and second order moments are given as:

$$M_1(dU_H) = \frac{1}{d}\delta_{i1i1'}\delta_{i2i2'} \tag{2.1}$$

and

$$M_2(dU_H) = \frac{(\delta_{i1i1'}\delta_{i2i2'}\delta_{j1j1'}\delta_{j2j2'} + \delta_{i1i2'}\delta_{i2i1'}\delta_{j1j2'}\delta_{j2j1'})}{d^2-1} - \tag{2.2}$$

$$\frac{(\delta_{i1i1'}\delta_{i2i2'}\delta_{j1j2'}\delta_{j2j1'} + \delta_{i1i2'}\delta_{i2i1'}\delta_{j1j1'}\delta_{j2j2'})}{d(d^2-1)}, \tag{2.3}$$

A circuit is said to exhibit a t-design if it matches the the Haar random distribution upto t-moments. Consequently, a two-design arises when the circuit matches the Haar distribution upto the first two moments as in Equation 2.1 and 2.2. For a sequence of parameterized unitary gates expressed as $U(\vec{\theta}) = \prod_{i=1}^{L} U(\theta_1)\dots U(\theta_L)$, we can denote:

$$U_- = \prod_{i=1}^{k} U(\theta_1)\dots U(\theta_{k-1}) \qquad U_+ = \prod_{i=1}^{k} U(\theta_k)\dots U(\theta_L)$$

for the k^{th} unitary U_k

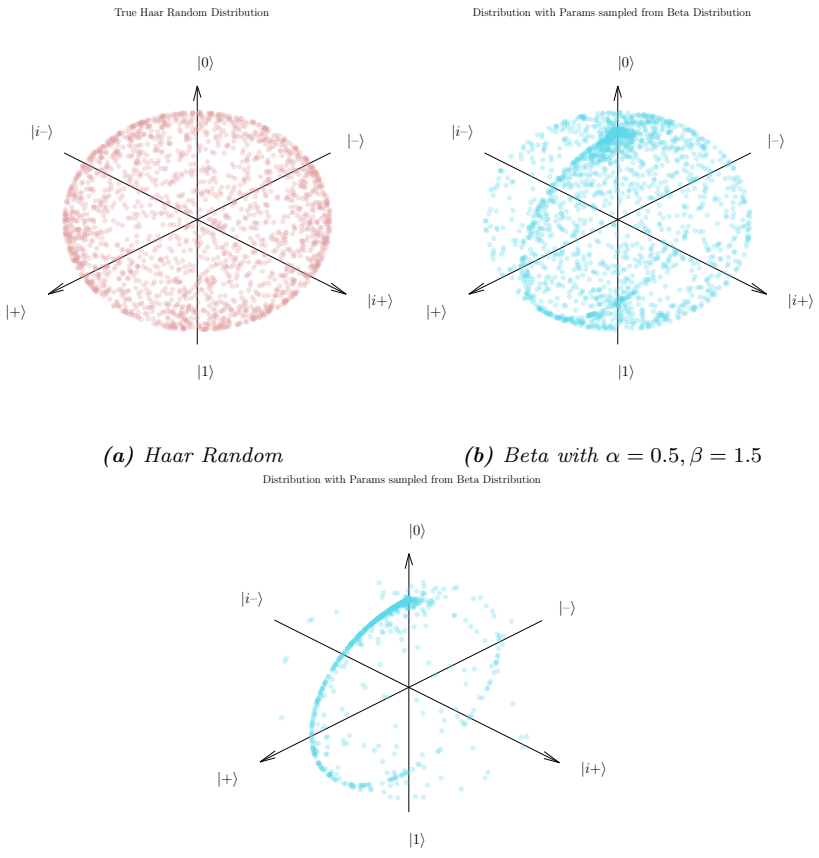
In[11] the authors show that if any U_- or U_+ exhibits a unitary 2-design then the variance of the classifier with respect to θ_k, $Var[\partial_k f]$ can be given by:

$$Var[\partial_k f] \propto \left(\frac{Tr(U^2)}{2^{3n}} - \frac{Tr(U)^2}{2^{4n}} \right) \tag{2.4}$$

In Equation (2.4), $U \in U(d)$ and n is the number of qubits. The relation assumes that both U_- and U_+ exhibit a 2-design and shows that with increasing number of qubits, the variance of the gradient will approach zero and consequently the QNN will be stuck in barren plateau with all conditions kept the same (other cases also exhibit a similar kind of decay).

2.2 The Effect of Initializing Distributions

A parameterized quantum circuit is represents a sampling from a unitary group $U(d)$ with the choice being guided by the initializing distribution. Thus, any change in the initializing distribution must also lead to a change in the way unitaries are sampled from the unitary group. This observation led us to perform an experiment where we rotated a base state $|0\rangle$ with a parameterized unitary $U(\theta, \phi, \omega)$ such that $|\psi\rangle = U(\theta, \phi, \omega)|0\rangle$. The parameters θ, ϕ, ω were independently drawn from a different distributions. We performed this experiment with $N = 21000$ independent samples with a uniform and beta distribution. To make the uniform distribution truly Haar Random (i.e. maximally distributed over the entire Hilbert Space $\mathbb{C}^2$) we modified $\theta' = sin\theta$. In the case of the Beta distribution we experimented with different values of hyperparameters α, β. The results of our experiment are shown Figure 2.1. It is very clear that changing the



(a) Haar Random *(b) Beta with* $\alpha = 0.5, \beta = 1.5$

(c) Beta with $\alpha = 0.1, \beta = 1.5$

Figure 2.1. *Quantum state plotted on a Bloch sphere after a parameterized unitary matrix was applied to* $|0\rangle$ *and the parameters were drawn from different distributions.*

initializing distribution *directly* leads to a change in the distribution over the states over the Hilbert space. Inspired by this we repeated the experiment in [11] with different distributions and profiled the variance of the gradient with the system size. The results are shown in Figure 2.2

2.3 The BEINIT Algorithm

We propose an algorithm termed BEINIT that is based on two key observations. First, drawing samples from a beta distribution for parametrizing the quantum circuit yields a much higher gradient variance for increasing number of qubits and second, it is possible to estimate the hyperparameters of the beta distribution from the data itself.

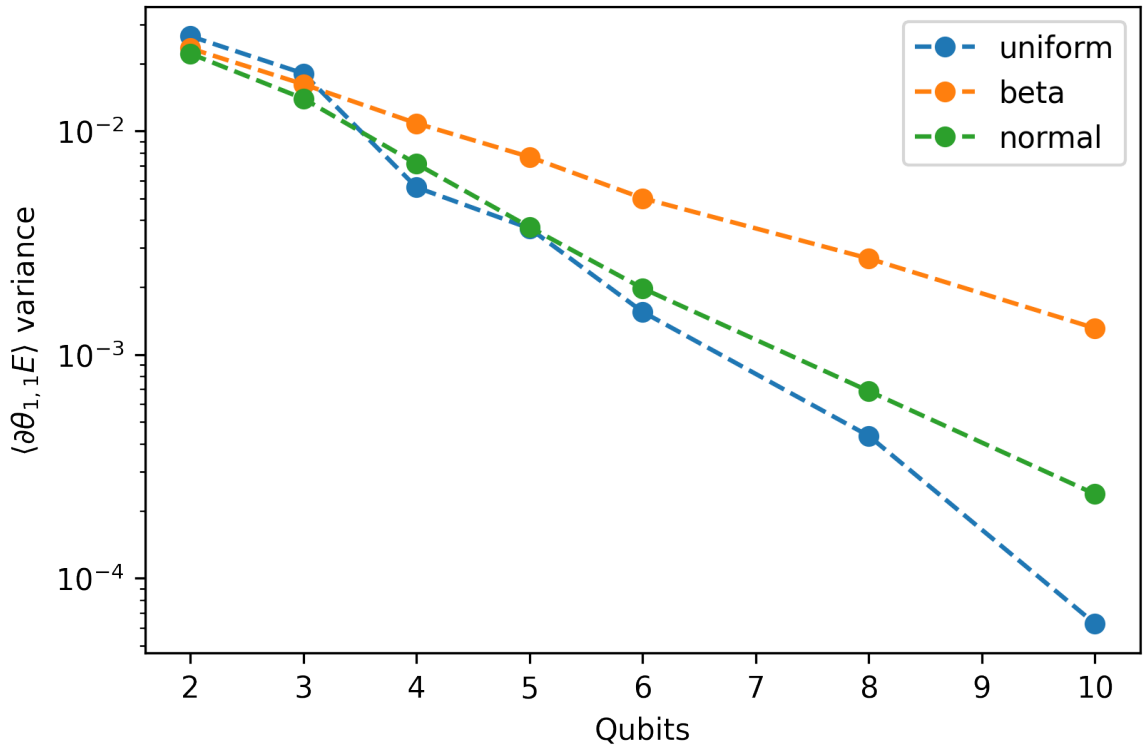


Figure 2.2. *Profiling Variance of the Gradient with different initializing distributions. The Beta distribution is significantly slower in degrading with all other conditions being kept the same.*

Algorithm 1 The BEINIT Algorithm

procedure BEINIT($\mathcal{D}, \eta, C(\theta), iters$)
  $\alpha, \beta \leftarrow$ EB_FIT($\mathcal{D}$)
  $\mathcal{D}_{train}, \mathcal{D}_{test} \leftarrow$ SPLIT($\mathcal{D}$)
  $\theta_i \leftarrow$ INIT(α, β)
  $\theta \leftarrow \theta_i$
  for $i = 0 \rightarrow iters$ **do**
    $\sigma_i^2 \leftarrow$ VAR($\nabla_\theta L$)
    $\sigma_n^2 \leftarrow \dfrac{\eta}{(1+i)^{\gamma+\sigma_i^2}}$
    $\theta' \leftarrow \theta + \mathcal{N}(0, \sigma_n^2)$
    $\theta^{i+1} \leftarrow$ GRAD_DESCENT($\theta', \mathcal{D}_{train}, \eta, C(\theta)$)

The second observation is based on the empirical Bayes framework.

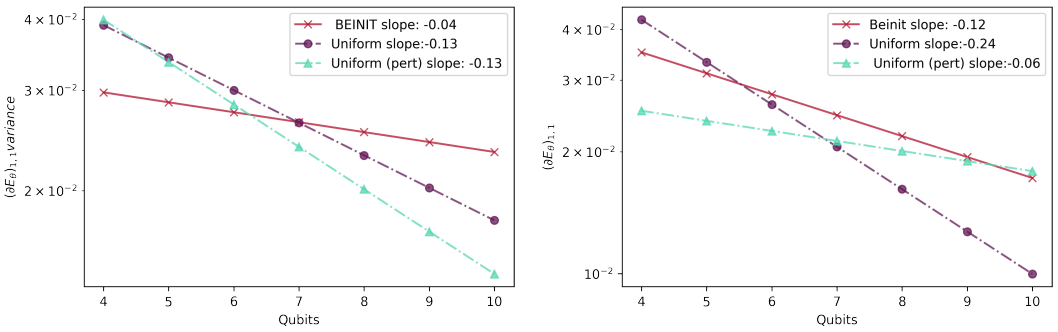
In this algorithm, we assume that we have access to a real valued dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^m$. For any given dataset $\mathcal{D}$ and a model with parameters $\boldsymbol{\theta}$, the goal of Bayesian machine learning is to determine the posterior distribution $\pi(\boldsymbol{\theta}|\mathcal{D})$. A prior distribution over the parameters is given by $\pi(\boldsymbol{\theta})$ and the likelihood function is given as $L(\mathcal{D};\boldsymbol{\theta}) = \prod_{i=1}^m p(x_i|\boldsymbol{\theta})$ where $x_i \in \mathcal{D}$. By the Bayes Law $\pi(\boldsymbol{\theta}|\mathcal{D}) \propto L(\mathcal{D};\boldsymbol{\theta})\pi(\theta)$. In a purely Bayesian model, we can set the prior $p(\boldsymbol{\theta})$ distribution to be a *known* parametrized distribution. We can further assume a *prior* on the hyperparameters of the parametrized distribution as well. However, doing this in practice is intractable for large dimensions or data points. A technique proposed in [15,16] alleviates this problem by estimating the hyperpameters from the data using Maximum Likelihood Estimation (MLE) approach. These initial hyperparameters are then used to draw samples from the prior distribution. We use this insight to impose a known prior distribution over the parameters of the unitary gates and estimate the shape parameters directly from the data.

For the Beta distribution the MLE estimation of α and β can be found as:

$$\alpha_{MLE} \equiv m\frac{\Gamma'(\alpha+\beta)}{\Gamma(\alpha+\beta)} - m\frac{\Gamma'(\alpha)}{\Gamma(\alpha)} + \sum_{i=1}^{m} x_i = 0 \tag{2.5}$$

$$\beta_{MLE} \equiv m\frac{\Gamma'(\alpha+\beta)}{\Gamma(\alpha+\beta)} - m\frac{\Gamma'(\beta)}{\Gamma(\beta)} + \sum_{i=1}^{m}(1-x_i) = 0 \tag{2.6}$$

Algorithm 1 describes the overall procedure. We take as input a labelled dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^m$, the learning rate η, a cost function $C(\theta)$ and the number of iteration steps *iters* as a required input. In the first step, we estimate the α and β parameters from all the available data using Equations (2.5) and (2.6). We then split this dataset into training and test subsets and initialize the circuit parameters from the discovered beta distribution parameters. During the training, we keep track of the gradient variance produced during an iteration i; σ_i^2. It has been shown that adding a perturbation during gradient descent can help a learning algorithm escape saddle points [17,18]. Inspired by a



(a) Iris *(b) Wine*

Figure 2.3. *The variance of the gradient of the first parameter is shown with increasing qubit sizes for three different initialization scenarios. Initialization with a beta distribution and added perturbation show the least degradation in variance with increasing qubit size.*

perturbation method proposed for deep learning[19], we introduce noise in the parameter space by considering a normal distribution parameterized by 0 mean and σ_n^2 variance given by:

$$\sigma_n^2 = \frac{\eta}{(1+i)^{(\gamma+\sigma_i^2)}}, \tag{2.7}$$

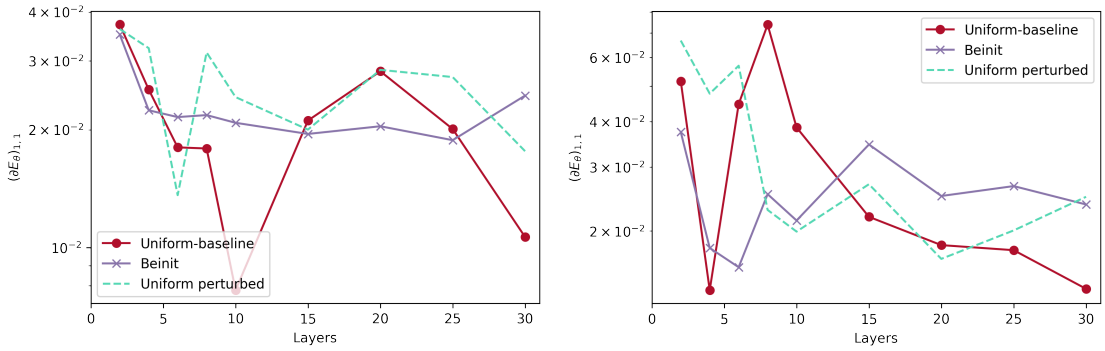
where η and γ are fixed parameters. One can interpret γ as a constant additive bias that prevents a low noise perturbation (since σ_i^2 progressively decreases as the output of the circuit gets closer to the true label). Empirically, the choice of η defines the amount of noise that will be generated per gradient descent step. We follow earlier works and choose $\eta \in \{0.01, 0.3, 1.0\}$ and keep $\gamma = 0.55$. It must be noted that our method generates this perturbation in the *parameter-space* as opposed the gradient space which is proposed by earlier related work[19].

2.4 Empirical Results

We demonstrate the effectiveness of our algorithm on the Iris and Wine datasets[20]. To study the effect of the BEINIT algorithm on the gradient variance with increasing number of qubits, we compare against two baseline cases. In the first case, we initialize the QNN's parameters using the uniform distribution as per[11] with the parameters estimated from the normalized data distribution. In the second case, we keep the same initialization as before, but introduce the perturbation as in Equation (2.7).

Both datasets considered in this study are multi-class classification problems, but we convert them into a binary classification problem by taking only first two classes. The labels are then binarized to ± 1. In the case of the Wine dataset, we reduce the dimensionality of the data by performing a principal component analysis (PCA) such that $d = 2$, where d indicates the dimensionality of a given data vector. We choose the number of qubits $q \in \{4, 5, 6, 7, 8, 9, 10\}$. All our experiments are performed on the default quantum simulator provided by the Pennylane[21] library and we use the Nesterov momentum optimizer[22] for all experiments. Figure 2.3 shows an interpolated line computed from the raw gradient values for different cases. We can observe that for both datasets, initializing with a beta distribution proves to be beneficial in reducing the likelihood of a 2-design from forming. We can also see that in certain cases (e.g. Wine dataset), perturbing the gradient even with uniform distribution can have a beneficial effect on the variance of the gradient. It is interesting to note that in both datasets, a variational quantum circuit initialized with uniform distribution exhibits a similar decay in variance as with a random circuit of[11]. This strongly indicates that a VQCs initialized with a uniform distribution have a higher likelihood of exhibiting a unitary 2-design.

Experiments with Deeper Circuits



(a) Iris *(b) Wine*

Figure 2.4. *The variance of gradient of first parameter is shown with increasing number of layers. The uniform distribution with no perturbation shows a downward trend with increasing layers. Beta and Uniform initialization with parameter perturbation result in a much more stable variance with beta initialization outperforming the uniform case.*

We also performed experiments with QNNs that were several layers deep on the two datasets above. In this set of experiments, we kept the number of qubits to be constant at 4 and varied the number of layers $L \in \{2, 4, 6, 8, 10, 15, 20, 25, 30\}$. Besides these changes, we kept the experimental setup similar to the qubit experiments before.
The variance of gradient of the first parameters in different cases is shown in Figure 2.4 shows the result of our experiments. As expected, the uniform distribution with no perturbation shows a poor scaling with increasing number of layers. With parameter perturbation, both uniform and beta initialization show a more stable behavior. In the case of the Iris dataset, the BEINIT procedure results in a steadier variance as compared to the uniform initialization with perturbation. The performance of BEINIT is also evidenced in the case of the Wine dataset. The results of these experiments shed light on the efficacy of both perturbation and data driven beta initialization for deep QNN circuits.

2.5 Connecting Parameter Initialization with VQC Expressivity

The BEINIT algorithm focuses on a case where p_{data} is fixed and the hyperparameters α, β can be estimated with a high degree of confidence. In this section, we explore a deeper connection of parameter initialization with the properties of quantum circuits. Specifically, we are interested in quantifying the deviation from the Haar measure when a particular initializing distribution is chosen to initialize the parameters of a VQC.

2.5.1 Parameter Initialization and Expressiveness

In order to connect parameter initialization with expressiveness, we define the following superoperator:

$$\mathcal{A}^{(t)}(.) = \mathcal{E}^{(t)}_{\mathbb{G}}(.) - \mathcal{E}^{(t)}_{\mathcal{U}}(.) \tag{2.8}$$

A superoperator is a linear map from one operator space to another. In other words, the action of a superoperator on a given linear operator yields another operator in the output space. The norm of the superoperator defined above i.e. $||\mathcal{A}^{(t)}(.)||_2^2$ gives the

deviation from the Haar distributed unitary ensemble and hence helps in quantifying the set of possible states that can be reached by a given quantum circuit.

We now define the individual terms in the right hand side of the Equation 2.8. To simplify exposition, we consider an input state ρ as:

$$\rho = \begin{pmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{pmatrix}$$

The first term on right hand side can be expanded as:

$$\mathcal{E}_{\mathbb{G}}^{(t)}(\rho) = \int_{\mathbb{G}} d\mu U^{\otimes t} \rho (U^{\dagger})^{\otimes t} \tag{2.9}$$

Where $d\mu$ is the measure associated with group $\mathbb{G}$. This integral can be easily calculated when $\mathbb{G}$ corresponds to $\mathbb{U}(d)$ i.e. a uniform distribution over unitary group of d-dimension and $d\mu$ is the Haar measure over the group.

In order to expand the second term we consider a parameterized VQC with parameters $\boldsymbol{\theta}$ and the unitary representation of the circuit as $U(\boldsymbol{\theta})$. We further assume that the parameters are initialized from a distribution $P_{\gamma}(\theta)$ where γ corresponds to the hyperparameters of the given distribution. Upon expansion, the second term reads:

$$\begin{aligned} \mathcal{E}_{\mathcal{U}}^{(t)}(\rho) &= \int_{\mathcal{U}} dU(\boldsymbol{\theta}) U(\boldsymbol{\theta})^{\otimes t} \rho (U(\boldsymbol{\theta})^{\dagger})^{\otimes t} \\ &= \prod_{l=1}^{L} \int_{\Omega_l} d\theta_l P_{\vec{\gamma}_l}(\theta_l) U(\boldsymbol{\theta})^{\otimes t} (\,\rho\,) (U^{\dagger}(\boldsymbol{\theta}))^{\otimes t} \end{aligned} \tag{2.10}$$

Where Ω_l is a parameter domain with associated probability $P_{\gamma}(\theta_l)$. Combining the second equality in Equation 2.10 and Equation 2.8 gives the explicit dependence of circuit expressiveness with the initializing distribution. An important consequence of this connection is that it allows us to formulate the theory behind the observations in the BEINIT algorithm.

2.5.2 VQC Expressiveness with Different Distributions

We now quantify the expressiveness of a quantum circuit when the parameters are initialized from two different distributions viz. the Uniform distribution $Unif(0, 2\pi)$

and the Beta distribution $Beta(\alpha, \beta)$.

To understand the mechanics, we first consider the case when a single parameterized gate is applied to an input state $|\psi\rangle$:

$$U(\theta, \phi, \omega) = e^{-i\theta Z/2} e^{-i\phi Y/2} e^{-i\omega Z/2}$$

Where $Z = \sigma_z$ and $Y = \sigma_y$ correspond to the elements of the Pauli group. For $t = 1$, we can show (via Weingarten Calculus, See Eq. 2.1 - 2.3):

$$\mathcal{E}^{(1)}_{\mathbb{U}(d)}(\rho) = \frac{1}{2} Tr[\rho] \mathbb{1} \tag{2.11}$$

For the uniform distribution with the volume element is given by $dU(\theta) = \frac{1}{4\pi^3} d\theta d\phi d\omega$. Using Equation 2.10, we analytically evaluate:

$$\begin{aligned} \mathcal{E}^{(1)}_{Unif}(\rho) &= \frac{1}{4\pi^3} \int_0^{2\pi} U(\theta, \phi, \omega) \rho U^\dagger(\theta, \phi, \omega) d\theta d\phi d\omega \\ &= \frac{1}{2} Tr[\rho] \mathbb{1} \end{aligned} \tag{2.12}$$

Similarly, for Beta we note the volume element $dU = \frac{1}{4\pi^3} Beta(\theta/4\pi^3, \phi, \omega)$. Carrying out the computation in same fashion, we arrive at:

$$\mathcal{E}^{(1)}_{Beta}(\rho) = \frac{1}{2} Tr[\rho] \mathbb{1} + c(\alpha, \beta) Tr[Z\rho] Z \tag{2.13}$$

Where $c(\alpha, \beta) = {}_2F_3(\{\alpha/2 + 1/2; \alpha/2\}, \{1/2, \alpha/2 + \beta/2, \alpha/2 + \beta/2 + 1/2\}; -\pi^2/4)$ and Z is the Pauli-Z matrix. We can use Equation 2.8 to compute the expressiveness of the circuit:

$$||\mathcal{A}^{(1)}_{unif}(\rho)||_2^2 = 0 \tag{2.14}$$

$$||\mathcal{A}^{(1)}_{beta}(\rho)||_2^2 = c(\alpha, \beta)^2 Tr[Z\rho] \tag{2.15}$$

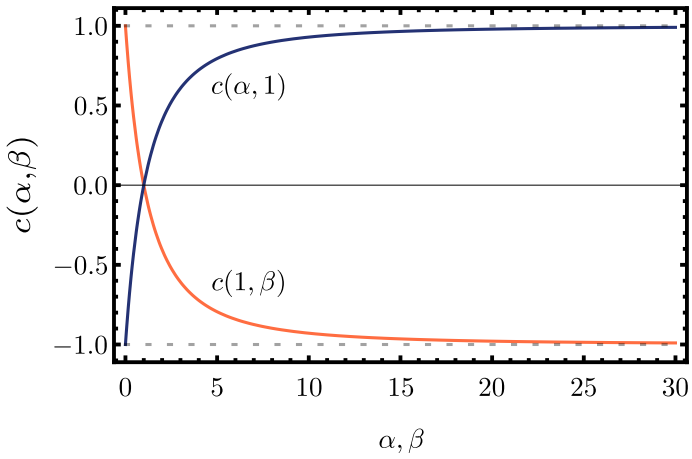


Figure 2.5. *The values taken by the coefficient $c(\alpha,\beta)$ for different values of α,β.*

A key observation that can be immediately made is that with a uniform distribution, the circuit exhibits a $t = 1$ design and will result in solution space similar to the Haar random distribution. From earlier discussion about barren plateaus, it is easy to see that a circuit initialized with the uniform distribution will get into barren plateaus even at shallow depths. On the other hand, using the Beta distribution results in a non zero expressiveness which indicates that the solution space is different than the Haar random distributed states. Figure 2.5 shows the value of the coefficient for different α,β. We can see $c(\alpha,\beta) = 0$ only when $\alpha = \beta = 1$. This case corresponds to the uniform distribution and hence our results are consistent in themselves.

Extending results to n-qubit and L-layered ansatz: We now extend the results obtained above for a single gate to the case of an n-qubit and L-layered ansatz. First, let's consider the case for a single qubit and L-layered graph.

We can express a parameterized circuit's twirl as:

$$\mathcal{E}^{(1)}_{p(\theta)}(\rho) = \prod_{l=1}^{L} \int dU_l U \rho U^\dagger \tag{2.16}$$

Where dU_l is the volume element associated with a given layers parameterized gates. The quantity $U\rho U^\dagger$ has a recursive nature:

$$U\rho U^\dagger = \prod_{l=1}^{L} U_l \rho (\prod_{l=1}^{L} U_l)^\dagger = U_1 \rho_{L-1} U_1^\dagger$$

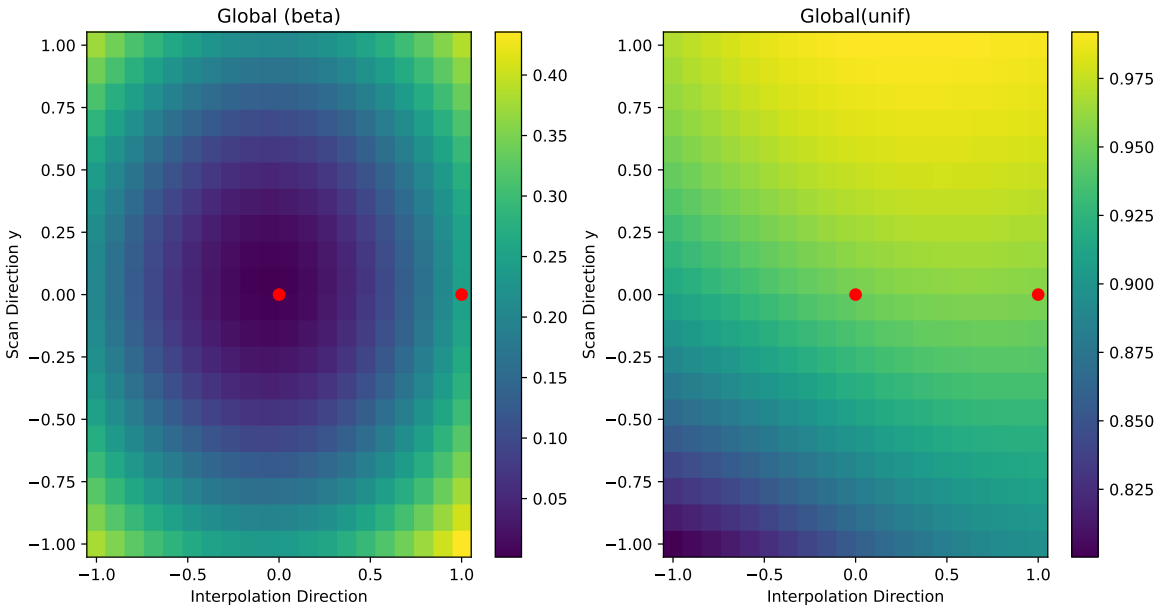


Figure 2.6. *Cost Landscape of a global cost function with parameters initialized with Uniform and Beta Distributions.*

And ρ_{L-1} is defined as:

$$\rho_{L-1} = \prod_{l=2}^{L} U_l \rho (\prod_{l=2}^{L} U_l^\dagger)$$

Thus, we can apply the results obtained in Equations 2.12 and Equations 2.13 recursively to obtain the expressiveness for the L-layered ansatz. For the uniform distribution, $\mathcal{E}^{(1)}_{unif}(\rho) = (\frac{Tr[\rho]}{2}\mathbb{1})$. For the Beta distribution, we obtain:

$$\mathcal{E}^{(1)}_{Beta}(\rho) = \frac{Tr[\rho]}{2}\mathbb{1} + \frac{c(\alpha,\beta)^{L+1} Tr[Z\rho]}{2} Z \tag{2.17}$$

Next, for a n qubit and L layered ansatz, $\mathcal{E}^{(1)}_{unif}(\rho)$ does not change. The twirl with Beta distribution can be calculated as:

$$\mathcal{E}^{(1)}_{Beta}(\rho) = \frac{1}{2^n} \sum_{q \in \{0,1\}^{\otimes n}} c(\alpha,\beta)^{w(\boldsymbol{q})(L+1)} Tr[Z^q \rho] Z^q \tag{2.18}$$

Where $w(\boldsymbol{q})$ is the Hamming weight over the bitstring $\boldsymbol{q} = Z_1^{q_1} \otimes \dots Z_n^{q_n}$. The expressiveness of these cases can be calculated by using Equation 2.8. Our motivation however, is to calculate the *depth* at which a unitary 1-design may occur for a given distribution. For the uniform distribution, the $||\mathcal{A}^{(1)}_{Unif}(\rho)||_2^2$ is zero. Hence, a circuit

initialized with the uniform distribution is always going to result in a 1-design. For the case of the Beta distribution, we can try and derive the bound for the case when an *approximate* 1-design occurs i.e. $||\mathcal{A}^{(1)}_{Beta}(\rho)||_2^2 \leq \varepsilon$. Treating $Tr[Z^q\rho] = \eta$, we can write $||\mathcal{A}^{(1)}_{Beta}(\rho)||_2^2$:

$$||\mathcal{A}^{(1)}_{Beta}(\rho)||_2^2 = \eta^2 \left(\frac{1 + c(\alpha,\beta)^{2(L+1)}}{2}\right)^n - \frac{\eta^2}{2^n} \tag{2.19}$$

Combining the inequality and Equation 2.19 and solving for L we get:

$$L \geq \frac{\log(\left(1 + \frac{2^n\varepsilon}{\eta^2}\right)^{\frac{1}{n}} - 1)}{2\log(c(\alpha,\beta))} - 1 \tag{2.20}$$

2.5.3 Numerical Results

We now provide numerical evidence of the change induced over the cost landscape seen by the VQC when initialized by different distributions. These figures were produced by utilizing the `orqviz`[23] package. In order to understand the plots we will define the notion of "global" and "local" cost functions.

Given the standard setup of VQCs, a cost function is defined as:

$$C(\boldsymbol{\theta}) = Tr[OU(\boldsymbol{\theta})|\psi_0\rangle\langle\psi_0|U^\dagger(\theta)] \tag{2.21}$$

Where O is the observable. A local and global cost function differs in the observable used for measurement. A global cost function has the observable:

$$O_G = 1 - |\mathbf{0}\rangle\langle\mathbf{0}| \tag{2.22}$$

This observable is equivalent to computing $1 - p(0)$ i.e. the probability of the output state being in the $|1\rangle$ state. A local observable is defined in[12] as:

$$O_L = \mathbb{1} - \frac{1}{n}\sum_{j=1}^{n} |0_j\rangle\langle 0_j| \otimes \mathbb{1}_j^- \tag{2.23}$$

Where $\mathbb{1}_j^-$ is identity on all qubits except the j^{th} qubit and n is the number of qubits in the circuit.

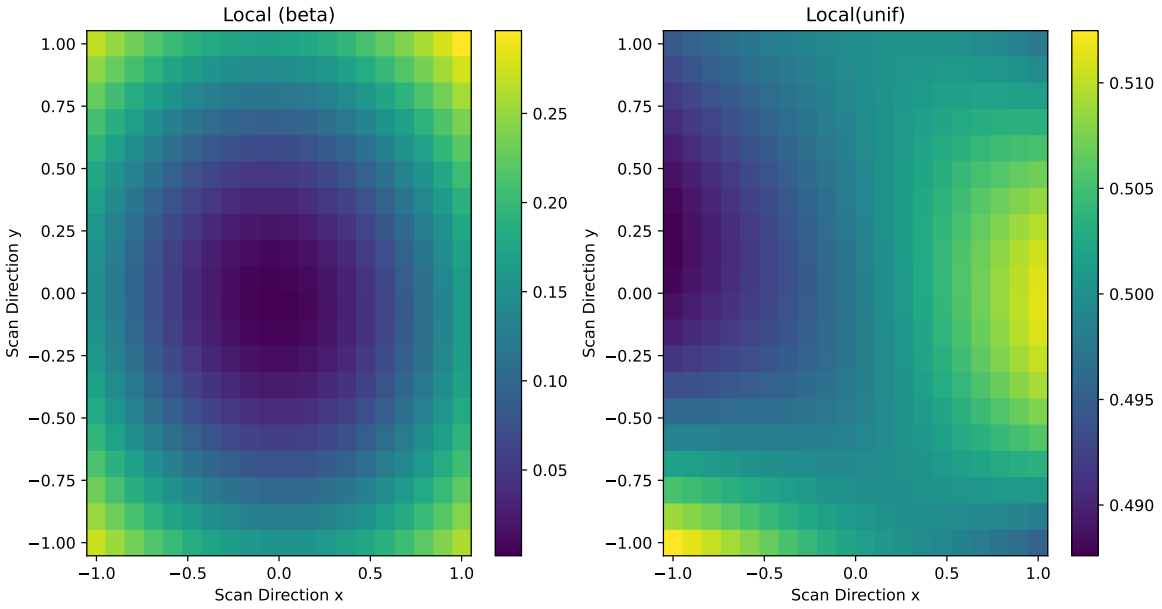


Figure 2.7. *Cost Landscape of a local cost function with parameters initialized with Uniform and Beta Distributions*

Interpolating the cost landscape: We compute an interpolated cost landscape for the given parameters $\boldsymbol{\theta}$ the following formula:

$$C_{interp}(\boldsymbol{\theta}) = C(\boldsymbol{\theta} + t_1\boldsymbol{d}_1 + t_2\boldsymbol{d}_2) \tag{2.24}$$

Where t_1, t_2 are interpolation coefficient sand $\boldsymbol{d}_1, \boldsymbol{d}_2$ are orthonormal directional vectors. The overall effect of this interpolation is a pilots "view" of the cost landscape when parameters are initialized from a specific distribution. Figure 2.6 shows the landscape for a global cost function and Figure 2.7 shows the landscape for the local cost function. In both cases we initialize a circuit with full entanglement with uniform and beta distributions.

In the case for both local and global cost functions, the Beta distribution shows a clear path towards a possible minima that can be attained by the circuit with the respective cost function. However, in the case of uniform distribution with a global cost function we see a "flat" landscape with no significant minima. With a local cost function, the situation improves slightly and the landscape appears to have atleast one minimum point. The conclusion from these plots is twofold. First, changing the initializing distribution drastically changes the cost landscape and second, any distribution other than uniform (e.g. Beta) can potentially help in creating a circuit that finds a better solution to the optimization problem.

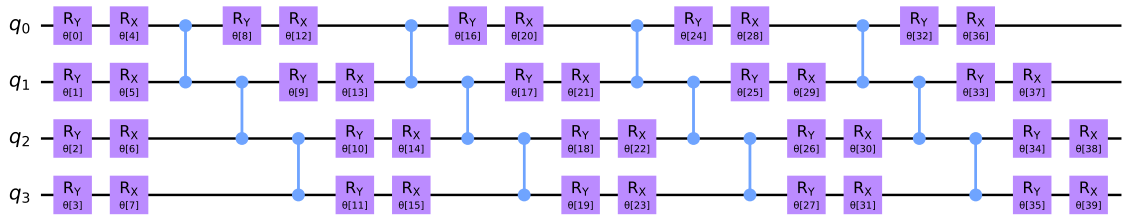


Figure 2.8. *The quantum circuit used in our experiments for measuring the trace norm of the unitary matrix when initialized with different distributions.*

Examining the Tracenorm of Unitary: To further study the changes caused by the parameter distribution, we compute the trace norm i.e. $\mathcal{T} = Re(Tr[U(\boldsymbol{\theta})]/2^n)$ for the unitary matrix representation of the circuit in Figure 2.8. The number of qubits are varied from $2 \rightarrow 9$ and the number of layers are held constant at 4. The parameters were separately initialized from Uniform ($\boldsymbol{\theta} \sim Unif(0, \pi)$) and Beta ($\boldsymbol{\theta} \sim Beta(\alpha, \beta)$) distributions.

Figure 2.9 shows the result of our experiments for different initialization strategies. When the parameters are initialized via the uniform distribution we note that $\mathcal{T}$ converges around zero very quickly. This implies that the quantum circuit quickly turns untrainable for high system sizes. In contrast, we notice that when the parameters are initialized using the beta distribution, the rate of convergence to around zero is much

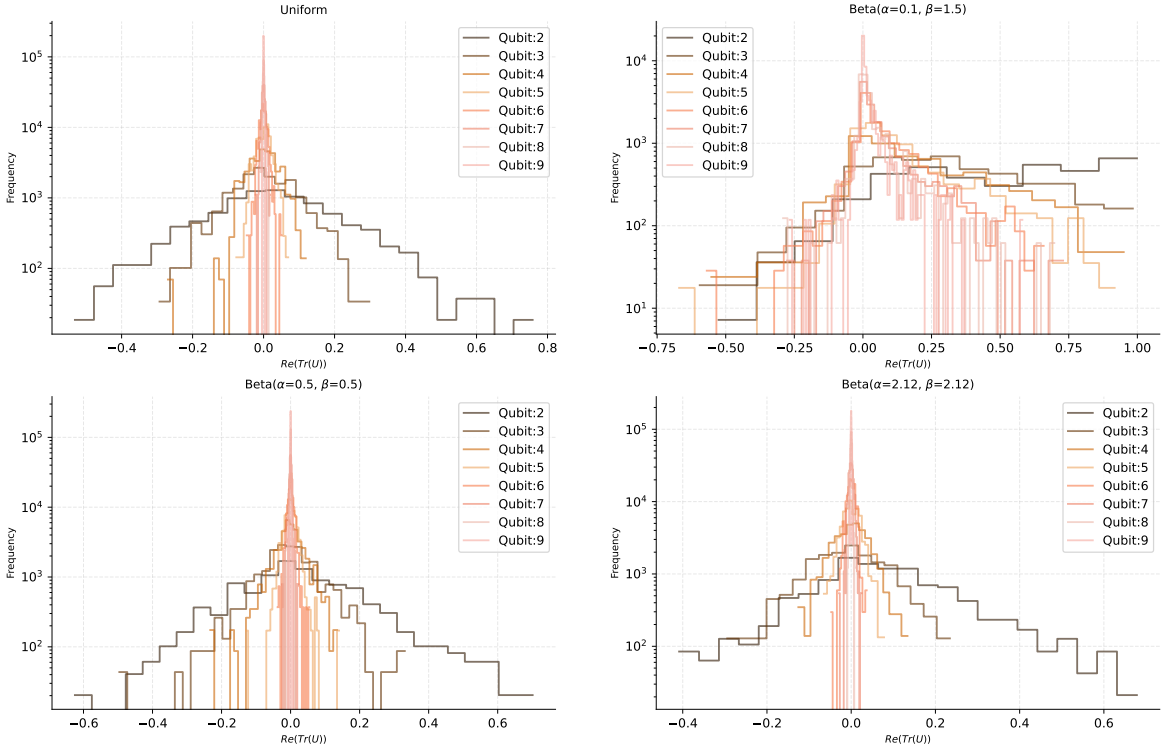


Figure 2.9. *Histogram showing the trace norm of the unitary matrix representation of the circuit in Figure 2.8 when initialized with uniform and beta distributions. The histograms are computed from 500 independent initializations from a given distribution.*

slower. This is especially true when $\alpha = 0.5, \beta = 1.5$ and $\alpha = 0.5, \beta = 0.5$. The reason for this slowness arises due to the presence of a hyperparameter dependent extra term in expressibility of a quantum circuit derived in earlier sections. We further consider the case when $\alpha, \beta > 1$ and notice that the rate of degradation is similar to uniform except that it occurs for slightly higher system sizes. An explanation for this behavior can be understood from the fact that when $\alpha = \beta = 1$ then the Beta distribution acts as a uniform distribution. Any value beyond that point for $\alpha = \beta$ resembles a *pseudo*-uniform distribution. A general heuristic about Beta initialization can also be drawn from the above figure - low values of α can help in preventing rapid degradation of a parameterized quantum circuit from becoming stuck in a barren plateau.

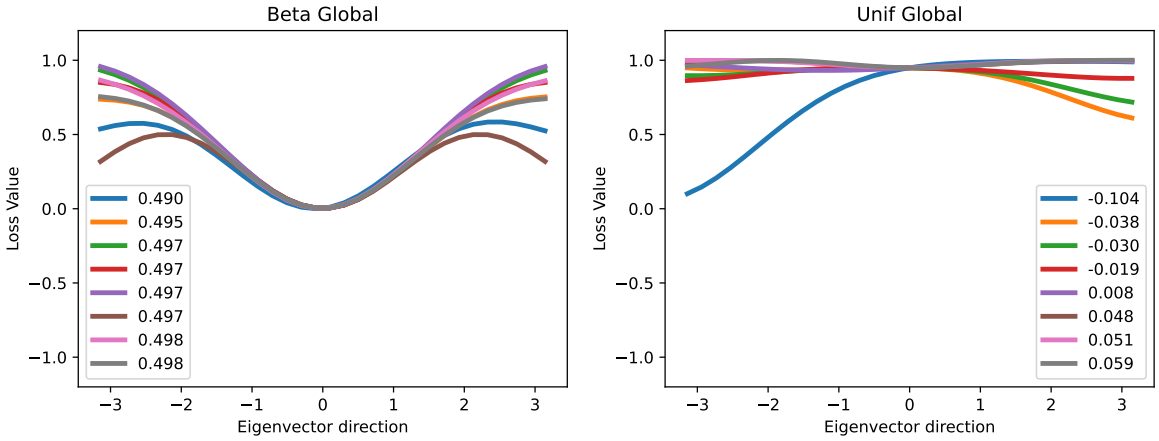


Figure 2.10. *Spectral decomposition of cost matrix with a global cost function for uniform and beta distributions.*

Eigenvalues of Hessian Matrix: The Hessian matrix is a central quantity in any gradient based optimization procedure and reveals several important details about the cost landscape for a given set of parameters and cost function. We focus on the spectral decomposition of the Hessian matrix:

$$\nabla^2 C(\boldsymbol{\theta}) = \Lambda \mathbb{E} \Lambda^T \tag{2.25}$$

Where Λ is the set of eigenvectors and $\mathbb{E}$ is the diagonal matrix consisting of the eigenvalues of the Hessian matrix. The eigenvalues describe the *curvature* of the cost landscape with the given set of parameters. A larger eigenvalue indicates a larger amount of curvature (and thus greater opportunity for finding a good minima). A 1-D interpolation of these values is plotted for global cost function in Figure 2.10 and for local cost function in Figure 2.11.

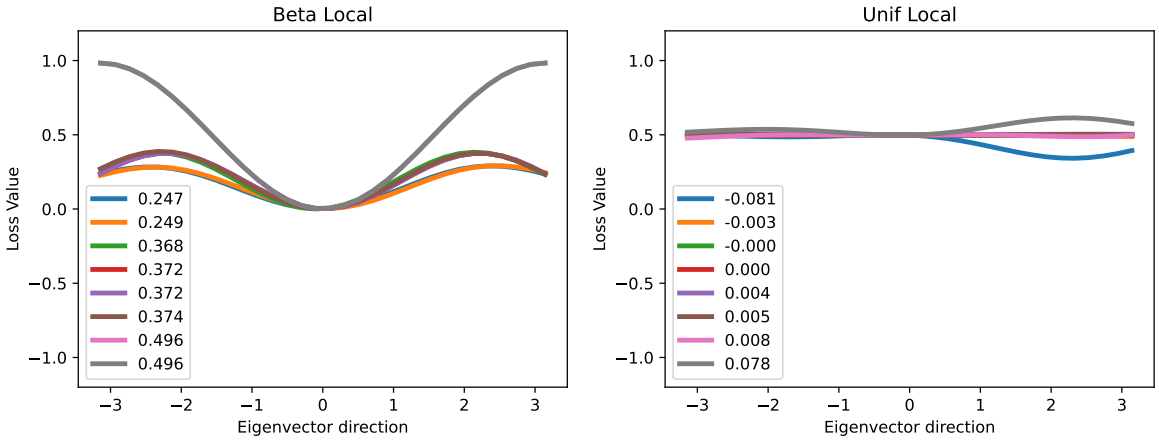


Figure 2.11. *Spectral decomposition of cost matrix with a local cost function for uniform and beta distributions.*

We can see that in the case of both global and local cost functions, the beta distribution yields higher positive eigenvalues and an interpolation of eigenvector directions shows a distinct curvature. In contrast, a uniform distribution has small positive eigenvalues and the interpolation shows a largely flat cost landscape.

Chapter 3

TRAINING VARIATIONAL CIRCUITS WITHOUT GRADIENTS

In this section we will discuss the second important direction of our work. This work arose in response to the second question posed in Chapter 1. We will introduce our method and demonstrate the theoretical and empirical speedup over conventional gradient based algorithms that currently exist for training VQCs.

3.1 Learning to Learn with LSTMs

We first dicuss the meta-optimization framework in more detail. For simplicity, we adopt a similar nomenclature as in[24] and refer to the QNN as the *optimizee* network and the meta-optimizer as the *optimizer* network.

In the meta-optimization framework, the task of the optimizee network is to evaluate the parameters suggested by the optimizer network. In turn, the optimization network accepts some input meta-features and performs an internal computation to generate new parameters for the optimizee network while adjusting its own parameters. In particular, if $\boldsymbol{\theta}^t$ are the optimizee parameters at time-step t during training, then the next parameters are discovered following[24]:

$$\begin{aligned} \boldsymbol{\theta}^{t+1} &= \boldsymbol{\theta}^t + \boldsymbol{g}^t\,, \qquad . \\ \begin{bmatrix} \boldsymbol{g}^t \\ \boldsymbol{h}^{t+1} \end{bmatrix} &= \mathcal{R}_{\boldsymbol{\Phi}}(\mathcal{I}(\boldsymbol{\theta}^t), \boldsymbol{h}^t) \end{aligned} \tag{3.1}$$

Here we specify $\mathcal{R}_{\boldsymbol{\Phi}}$ to be an instance of LSTM that accepts a hidden state $\boldsymbol{h}^t$ and the meta-parameters $\boldsymbol{\Phi}$ and outputs the update $\boldsymbol{g}^t$. The LSTM accepts $\mathcal{I}(\boldsymbol{\theta}^t)$ which we define to be a function that combines information from the training process and presents it as input features to the optimizer network. For instance, in[24] and similar

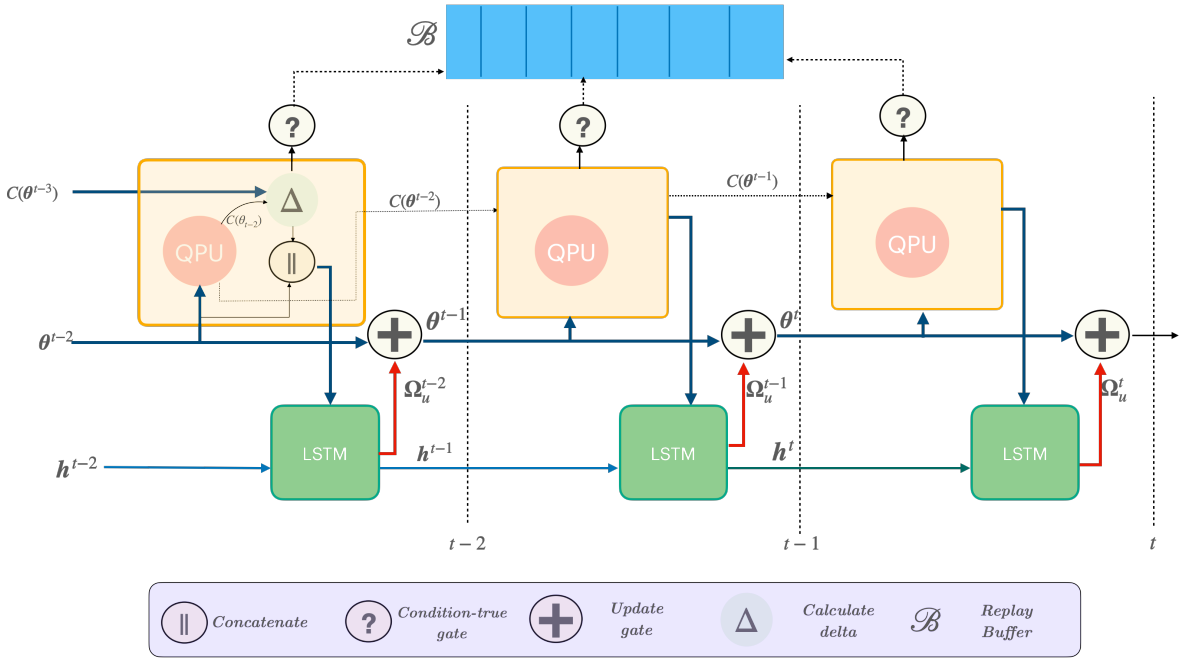


Figure 3.1. *An overview of our proposed meta-learning algorithm. The blue arrows indicate key inputs to the QNN and the LSTM and the red arrows indicate the output from LSTM. Best viewed in color.*

works[25,26], $\mathcal{I}(\boldsymbol{\theta}) = \nabla_\theta C(\boldsymbol{\theta})$. The general form of meta-loss function tries to minimize the loss over a finite horizon window of size T. In this work, we follow a meta-loss function inspired by the deep learning literature:

$$\mathcal{L}(\boldsymbol{\Phi}) = \sum_{i=1}^{T} w_t C(\boldsymbol{\theta}^t), \tag{3.2}$$

where $\boldsymbol{W}^m = [w_1, w_2, \ldots, w_T]$ are the weights over the cost function evaluations at time step t. A uniform weighting strategy sets all weights to be equal. The *magnitude* of the weights is a matter of choice and the decision is made based on the dataset that is being used. The optimization of $\boldsymbol{w}$ in a data driven manner is out of scope of this paper. In general, we suggest that a line of work based on hyper-parameter optimization[27] can be developed and used to dynamically adjust the weights for any given data.

3.1.1 Our Algorithm

We will now discuss the main components of our algorithm. Although, we generally follow the meta-optimization framework discussed above, the constraints induced by the need for gradient free optimization lead us to introduce the algorithmic tools that have not been proposed in literature in the meta-optimization context for quantum machine learning.

Input Preprocessing: In earlier works, the input function involved passing some gradient information from the optimizee network to the optimizer network. Since we are constrained to not use gradient information, we utilize the tuple $(\boldsymbol{\theta}^t, \Delta C(\boldsymbol{\theta}))$ as inputs where $\Delta C(\boldsymbol{\theta})$ is a *pseudo-gradient* that computes the difference between the current and previous cost function evaluation, i.e., $\Delta C(\boldsymbol{\theta}) = C(\boldsymbol{\theta}^{t-1}) - C(\boldsymbol{\theta}^t)$. These inputs are concatenated in a single vector and the input function $\mathcal{I}(\boldsymbol{\theta})$ is prepared as:

$$\mathcal{I}(\boldsymbol{\theta}) = \frac{\mathcal{P}([\boldsymbol{\theta}^t; \Delta C(\boldsymbol{\theta}^t)])}{p}. \tag{3.3}$$

Here we use a non-linear function $\mathcal{P} : \mathbb{R}^d \mapsto [0, 1]$ that normalizes the input values to between 0 and 1. The value p controls the strength of normalization. In our experiments we use $\mathcal{P}(\boldsymbol{x}) = e^{\boldsymbol{x}}$ with $\boldsymbol{x}$ being the input vector for the LSTM. We empirically found $p = 50$ to work best for the datasets considered in this study. In a future work, we would like to explore the possibility of learning an adaptive normalization strength that is derived from observing history of updates during training.

Non Linear Parameter Updates: In Equation (3.1), we defined a generic update rule in the meta-optimization framework. That update rule simply adds the output of LSTM to previously obtained parameters. In this work, we introduce a new update rule of the form:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \alpha \cdot \sigma(\boldsymbol{\Omega}_u^t), \tag{3.4}$$

where $\boldsymbol{\Omega}_u^t$ are the updated parameters obtained from the LSTM. Compared to Equation (3.1), we have changed $\boldsymbol{g}^t = \boldsymbol{\Omega}_u^t$ to $\boldsymbol{g}^t = \alpha \cdot \sigma(\boldsymbol{\Omega}_u^t)$ where α is a hyper parameter and σ is a non-linear activation function (tanh in our implementation). This update rule applies a non-linear activation to the LSTM parameters and controls the strength

of the update using α. Informally, α can be interpreted as a learning rate which helps the quantum learner in adjusting the new parameters. Since there is no direct gradient information, a destructive update (i.e., $\boldsymbol{\theta}^t = \boldsymbol{\Omega}_u^t$) would make the optimizee's parameters prone to oscillation due to the noisy output from LSTM. Additionally, the raw parameter updates from LSTM are unbounded in $\mathbb{R}$. A clipping function (e.g., tanh) that clips the values between certain maximum and minimum values, leads to more stable updates which consequently yield a smoother parameter update.

Replay Buffer: In earlier works, the LSTM network was able to compute a good descent direction based on gradient information provided to it during training. Even in such works as[28], the LSTM network was only used to learn the initial parameters for optimizing a QNN and then the conventional gradient descent method was used. *However, we consider a harder case where no gradient information is available.* It is already known that a short horizon bias problem exists for learned optimizers for classical neural networks[29]. In the case of optimizing quantum networks without using any gradient information, this problem becomes even harder to overcome.

In meta-optimization a short horizon bias occurs when the meta-optimizer becomes biased to providing updates akin to taking short steps towards minima. These updates do not cause a significant overall decrease in optimizee's cost function. This effect is more pronounced when we operate in parameter space as opposed to gradient space since the LSTM network does not get any information about the curvature of the optimization surface. Running optimization in a finite horizon window in parameter space can then cause LSTM to suggest incorrect updates to the optimizee network. In the specific case of QNNs, the parameters correspond to a rotation of given input state about a particular axis. An incorrect update can very easily cause the quantum state to be rotated incorrectly and therefore lead to an increase in the value of the objective function we're interested in minimizing. To overcome this issue, we develop a technique inspired by work in reinforcement learning[30]. Throughout training, we keep track of past history of parameters and their corresponding cost function values in a "replay buffer". At the start of training we instantiate a double ended queue dubbed

as a *replay-buffer* $\mathcal{B}$ of a finite capacity R. For meta-iteration $t = 1 \dots T$ we observe a history of parameters $\boldsymbol{\theta}^t$ and the corresponding cost $C(\boldsymbol{\theta}^t)$. If $C(\boldsymbol{\theta}^{t+1}) < C(\boldsymbol{\theta}^t)$ then we add the state $s = [\boldsymbol{\theta}^{t+1}, C(\boldsymbol{\theta}^{t+1}), \Delta C(\boldsymbol{\theta}), \boldsymbol{h}^{t+1}]$ to the replay buffer. Once the meta-iteration ends and $\mathcal{L}(\boldsymbol{\Phi})$ is computed, if the QNN cost function is diverging, we seed the parameters for the next meta-iteration by performing the following update:

$$\begin{aligned} \boldsymbol{\theta}^{T+1} &= \tau \cdot \boldsymbol{\theta}^T + (1 - \tau) \cdot \boldsymbol{\theta}_s \\ \tau &= \frac{\tau}{(1 + \zeta \cdot t)}. \end{aligned} \quad , \tag{3.5}$$

where ζ is a decay factor that adjusts the blending coefficient τ as the training progresses and $\boldsymbol{\theta}^T$ are the optimizee parameters at the end of a previous unrolled meta-iteration loop. $\boldsymbol{\theta}_s$ are the sampled parameters from the replay buffer that are chosen according to a fixed policy function $\pi(\mathcal{B})$. In our work this policy corresponds to choosing the parameters that lead to most cost decrease over the previous unroll iterations. We expect that future works in this directions will be able to learn $\pi(\mathcal{B})$ along with parameters. We set τ to be 0.9 to put more emphasis on the current parameters than sampled parameters. As training progresses and we observe divergent behavior, we decrease τ according to Equation 3.5 with $\zeta = 0.99$ and t indicating the overall global step of optimization iteration.

Overall Algorithm: The overall flow of the algorithm is shown in Figure 3.1. At a given timestep t, we expect the cost function value $C(\boldsymbol{\theta}^{t-2})$ and the parameters $\boldsymbol{\theta}^{t-1}$. We then compute the cost $C(\boldsymbol{\theta}^{t-1})$ using Equation (**??**). A cost delta is computed $\Delta C(\boldsymbol{\theta}) = C(\boldsymbol{\theta}^{t-2}) - C(\boldsymbol{\theta}^{t-1})$, which is then used as a decision metric and a feature in the input. In the former role, if $\Delta C(\boldsymbol{\theta}) < 0$ then a sample (described earlier) is committed to the replay buffer $\mathcal{B}$. Then, depending on the availability of elements in $\mathcal{B}$, the parameters are either sampled or retained from previous step. An input feature is then pre-processed using Equation (3.3). The updated parameters $\boldsymbol{\Omega}_u^t$ are then obtained from the LSTM and $\boldsymbol{\theta}^t$ is computed using Equation (3.4). In the initial conditions, we set $\Delta C(\boldsymbol{\theta}) = 1.0$

3.2 Theoretical Performance

We now analyze the theoretical performance gain that our algorithm can provide in the context of parameterized quantum circuits. We note that our algorithm is not predicated on an existence of an efficient data structure like QRAM[31], rather the performance gains are expected due to non-computation of gradient on quantum devices.

Theorem 1. *Consider a variational quantum circuit consisting of q qubits, L layers and k single qubit gates per layer. Let $\mathcal{A}$ be an optimization algorithm running on a classical device helping the circuit minimize a cost function $C(\theta)$. Then:*

- *If $\mathcal{A}$ is a gradient-dependent algorithm the the total time for one full pass (forward and backward) takes $O(2(qLk)^2\delta t_f)$, where δt_f is the time for a forward pass.*
- *If $\mathcal{A}$ does not require a gradient, then the total time for one full pass takes $O(2(qLk)\delta t_f)$.*

Proof. Consider an instance of gradient dependent algorithm $\mathcal{A}^g$. To update the parameters for the next step, it requires $\nabla_\theta C(\theta)$. For a circuit running on NISQ computer, currently the only method to compute gradients is to use the parameter shift method[32]. The expected gradient w.r.t single component θ_i is given as:

$$\nabla_{\theta_i} C(x;\theta_i) = \langle\psi_x|\nabla_{\theta_i}\mathcal{F}_{\theta_i}(\hat{\boldsymbol{O}})|\psi_x\rangle, \tag{3.6}$$

where $|\psi_x\rangle = \boldsymbol{U}_0(x)|0\rangle$. The quantity $\nabla_{\theta_i}\mathcal{F}_{\theta_i}(\hat{\boldsymbol{O}})$ is the gradient w.r.t to the i^{th} component of the parameter vector. The parameter shift rule states:

$$\begin{aligned} \nabla_{\theta_i}\mathcal{F}_{\theta_i}(\hat{\boldsymbol{O}}) &= c[\mathcal{F}_{\theta_i+s}(\hat{\boldsymbol{O}}) - \mathcal{F}_{\theta_i-s}(\hat{O}))] \\ \mathcal{F}_{\theta_i}(\hat{\boldsymbol{O}}) &= \boldsymbol{U}^\dagger(\theta_i)\hat{\boldsymbol{O}}\boldsymbol{U}(\theta_i). \end{aligned}, \tag{3.7}$$

where c is a scaling constant ($c = 0.5$) and s is a constant shift angle about which the gradient is computed. To successfully evaluate $\nabla_{\theta_i}C(\theta)$ one needs two evaluations of the quantum circuit with shifted parameters. Consequently, the time for one gradient computation over the entire quantum circuit is $O(2qlK)$. Then, for a full pass (i.e. computing the cost and gradient), the computation time for $\mathcal{A}^g$ scales as $O(2(qlK)^2\delta t_f)$. In contrast, a gradient free method (like ours), does not incur the penalty of computing the gradient and thus the overall computation time scales as $O(2(qlk)\delta t_f)$. □

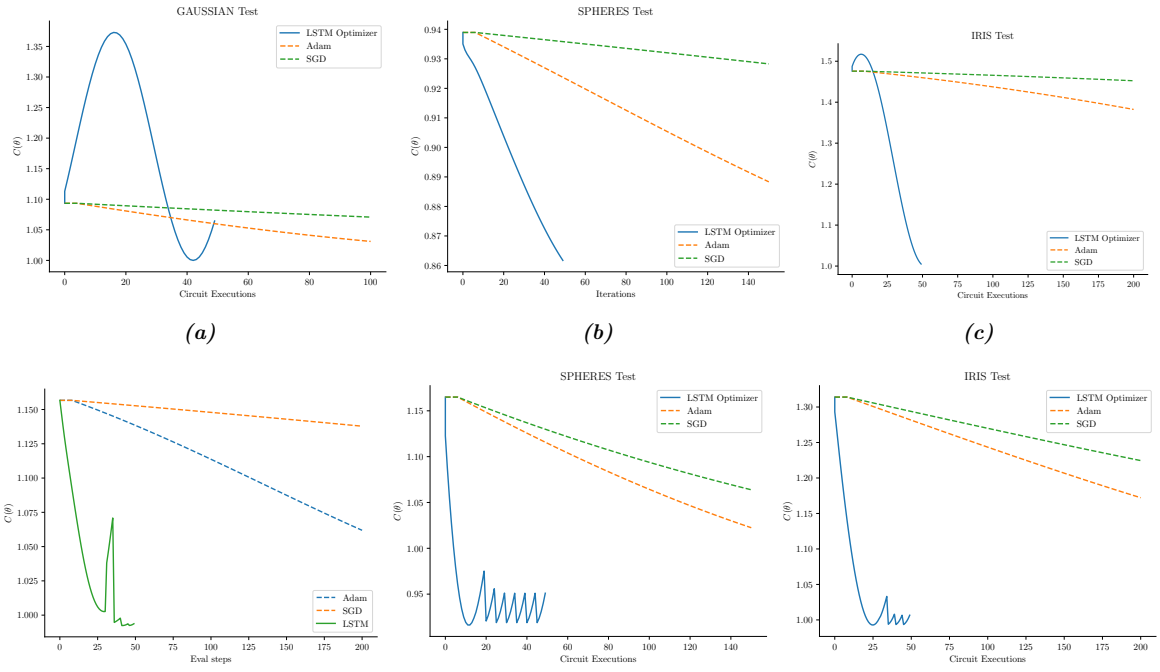


Figure 3.2. *Performance of LSTM-based meta-optimizer on the binary classification task for three datasets. All parameters are initialized using the normal distribution. The top row shows the performance of the algorithms without any replay buffer sampling and the bottom row shows the performance with replay buffer sampling. In both cases, the LSTM based optimizer is able to achieve a lower cost in significantly fewer circuit evaluations.*

The reduction of a quadratic run-time to a linear run-time is significant since this will allow a QNN to scale to a larger number of data points. Although, our method does incur a storage overhead of $O(R)$, for all practical scenarios $O(R) \ll O(qlK)$.

3.3 Empirical Results

3.3.1 Experiments on Machine Learning Datasets

We first present our experiments on a pattern classification task using an instance of a layered ansatz. The first layer in the ansatz embeds the input data $\boldsymbol{x}$ into the corresponding angles of a RX gate, where $RX(x_j) = e^{-ix_j\sigma_x/2}$ and σ_x is the Pauli-X matrix. The subsequent layers apply a parameterized RY rotation and are entangled using the CZ gate. Since the number of input qubits is equal to the dimensionality of data, we consider three datasets with increasing dimensionality - Gaussian, Spheres

and Iris. The Gaussian dataset is a synthetic dataset where two-dimensional clusters are instantiated by drawing samples from a multivariate Gaussian distribution of given parameters. Similarly, the spheres dataset is a collection of 3-d points which form concentric spheres with one sphere enveloping the other. Finally, we consider a truncated Iris dataset that consists of only two classes and four features that describe a particular species. In all these datasets, the labels are encoded into $\{-1, +1\}$ and the optimization task is to find variational parameters that minimize Equation **??** with Pauli-Z gate being the observable.

In the experiments we do not apply any pre-processing or post processing on the input data. We benchmark our LSTM optimizer against two commonly used gradient based algorithms - ADAM[33] and Gradient Descent. For the LSTM optimizer, the ansatz is run for 50 iterations while for gradient based algorithms we run it for 25 iterations. We then profile the cost obtained by the corresponding algorithms with the number of circuit evaluations. The learning rate for gradient based algorithms is set as $1e^{-2}$ and $\alpha = 0.1$ for the LSTM optimizer.

Figure 3.2 shows the results of our experiments on the three datasets. The top (bottom) rows show the performance without(with) replay buffer sampling. In all cases, the LSTM optimizer is able to find parameters that minimize the cost function in far fewer circuit evaluations than gradient based optimizers. This result is an empirical validation of Theorem 1 and intuitively makes sense since parameter shift rules evaluate the same circuit twice per parameter component to estimate the gradient at a given timestep.

Figure 3.2a shows a phenomenon unique to LSTM based optimizers. After finding a good descent direction, the LSTM based optimizers tend to over optimize and thus oscillate about some fixed point. This "sinusoidal" osciallation is hard to control in optimization problems since it's not easy to guess the stationary points in multivariate objective functions. Our replay buffer sampling strategy is aimed to control this phenomenon in a more principled manner and it's effects are visible in Figure 3.2d where the parameter mixing and decay lead to a more stabler convergence. The effects are

also visible in Figures 3.2e and Figures 3.2f where the cost function does not diverge after reaching a minimum point.

3.3.2 Comparison Against Other Gradient Free Approaches

We now benchmark our algorithm against another well known gradient-free algorithm - Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm[34] which approximates gradients by taking finite differences between perturbed parameter vectors. The perturbations are generated stochastically.

We perform experiments with a four qubit, five layer variational circuit whose minimum parameters correspond to the minima of the following cost function:

$$C(\boldsymbol{\theta}) = \bigotimes_{i=1}^{n} \langle \psi(\boldsymbol{\theta}) | \sigma_z(i) | \psi(\boldsymbol{\theta}) \rangle \tag{3.8}$$

Where $|\psi(\boldsymbol{\theta})\rangle$ is a variational circuit consisting of strongly entangling layers i.e. rotation gates with all-to-all entanglement and $\sigma_z(i)$ is the Pauli-Z matrix acting on the i^{th} qubit as the observable. The results of our experiments are shown in Figure 3.3. We can see that the LSTM Optimizer is able to find a better quality minima than SPSA as well. Moreover, SPSA makes an extra call to the circuit per optimization step making it slightly more expensive than LSTM optimizer. We conclude that the our gradient free algorithm can help VQCs scale to larger and more complex problem instances in the future.

3.3.3 Time Profiling Results

In our earlier experiments we noticed that LSTM optimizer converges to a minima in fewer circuit evaluations than other methods. This implied that our method could be used to scale to data with larger number of points. To test this hypothesis, we perform experiments for a pattern classification task by generating N samples from a Gaussian distribution of finite mean and covariance. We benchmarked the time per epoch (i.e. time it takes to go through an entire dataset) for our method against Adam, Gradient Descent Optimizer and RMSProp[35].

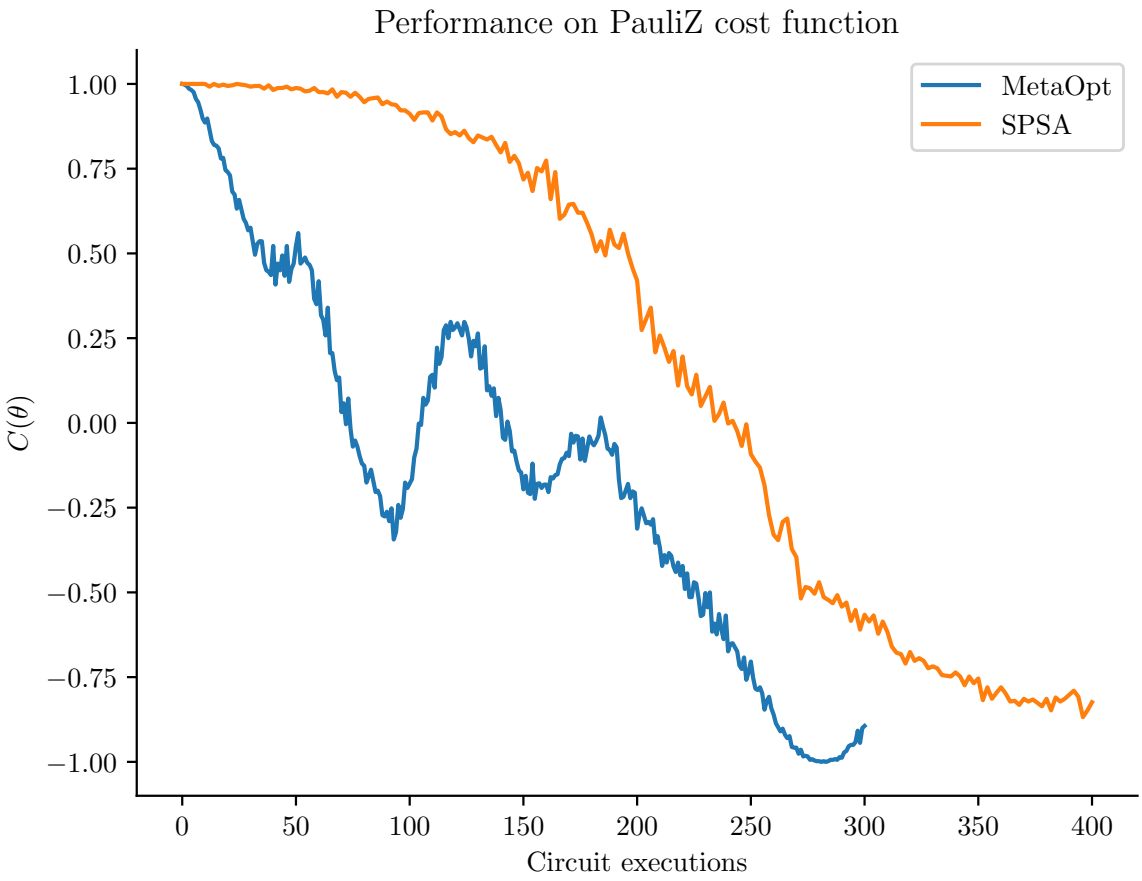


Figure 3.3. *Performance of LSTM based optimizer benchmarked against the SPSA algorithm. For the same cost function, SPSA makes an extra call to the quantum circuit which results in more number of circuit evaluations. LSTM based optimizer outperforms SPSA in terms of cost function minimum.*

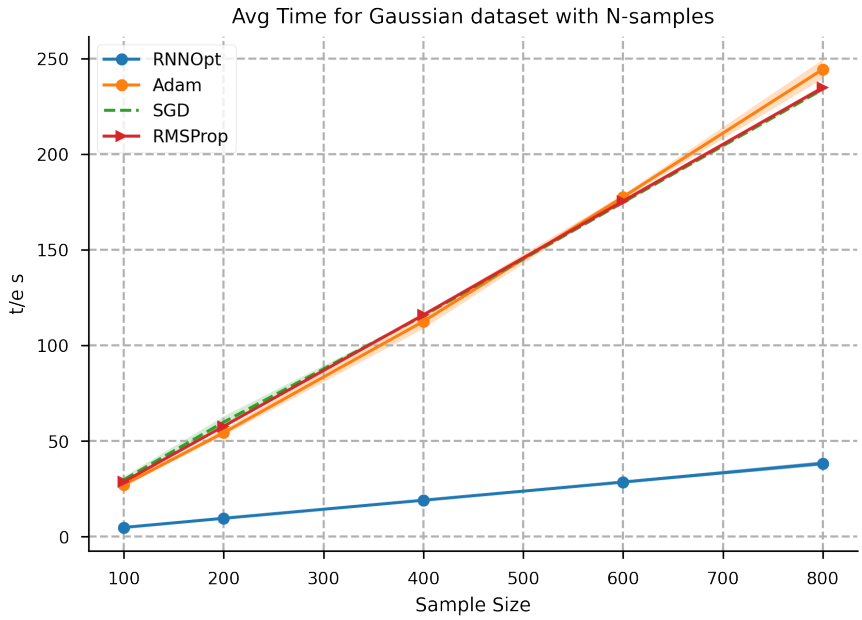


Figure 3.4. *Time profiling results for different sizes of Gaussian datasets.*

The results of our experiment are shown in Figure 3.4. We vary the sample size $N = \{100, 200, 400, 600, 800\}$ and measure the scaling of the time per epoch it takes for the optimizers. It is clear from the figure that the LSTM optimizer is able to scale to larger data instances without a significant increase in the time per epoch. In fact, our method is nearly *three* times faster than any gradient based method on the same dataset. This result is a positive indication that development of novel gradient free methods is a novel research direction and improvements may lead to VQCs being able to scale to handle data at the scale which is currently being handled by classical algorithms.

3.4 Conclusion and Future Work

This work introduced the idea that gradient free optimization is possible for quantum circuits and outperform existing gradient based algorithms in terms of the circuit calls needed to obtain the minima and often outperform these algorithm in terms of the actual minima obtained. We noted that this condition is critical towards achieving "quantum supremacy" and thus our work is a positive step towards that direction.

A future work direction is to apply this algorithm to solve problems in other VQA applications like Max-Cut QAOA[2], estimating the ground state energy of molecules[6]. We hope to observe a similar level of performance gain as has been observed above.

Chapter 4

PARAMETER PRUNING FOR BETTER TRAINABILITY

In this section we will explore a pruning algorithm to determine if it can enhance trainability of a quantum circuit by adaptively determining the irrelevant parameters during the course of optimization.

4.1 The QAdaPrune Algorithm

Theoretically, a VQC may be able to approximate an arbitrary function $f : \mathbb{C} \mapsto \mathbb{R}$ if infinite resources are available for simulation on a quantum device. However, in practice, the limited gate depth of current generation quantum computers and the inherent sensitivity of qubits to noise make it extremely hard to approximate such functions efficiently. This limitation has led to development of methods to produce low-resource circuits that can learn an optimal function f^* from a given family of all possible functions in Hilbert space (collectively called as the set of hypothesis functions $\mathcal{H}$)[36]. These methods fall in two broad categories. The first category consists of circuit level methods that include transpiling of a complex simulator circuit into an efficient gate level physical circuit[37], generating efficient quantum circuits from a known "supercircuit"[10]. The second category consists of methods that aim to reduce circuit complexity by employing some version of parameter pruning. For instance, Wang *et al.*[38] propose utilizing parameter shift rules[32] to evaluate gradient on chip and a probabilisitc gradient pruning algorithm to remove ineffective parameters. Hu *et al.*[39] propose a compression pipleine which involves alternate pruning and quantization steps to reduce the overall number of parameters in the circuit. Yet another method is introduced by Sukin *et al.*[40] where an online parameter pruning method is proposed. It adaptively prunes and grows parameters based on four input hyperparameters - the

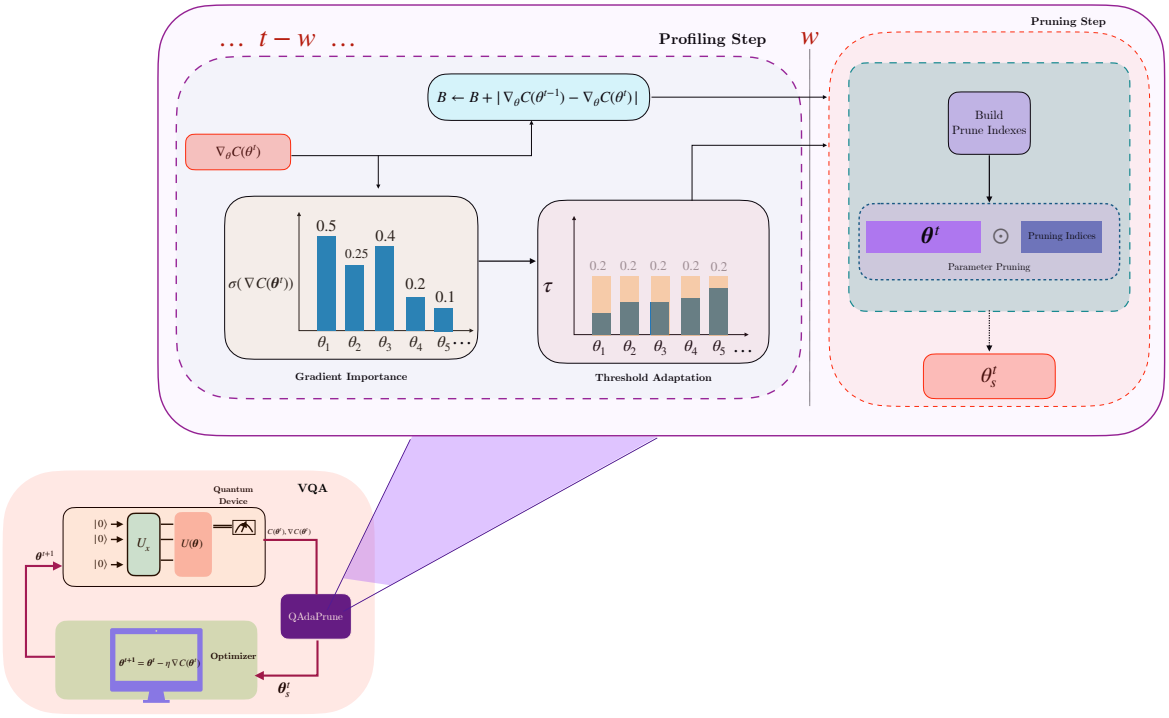


Figure 4.1. *Overall algorithmic flow of the QAdaPrune algorithm. Two key components of gradient importance and adaptive threshold are performed continuously until* $t = nw$*. When* $t \bmod nw = 0$*, a pruning index is built and the ineffective parameter components are pruned.*

global sparsity, number of parameters to prune at a given timestep, pruning threshold and tolerance value.

In contrast to neural networks where neurons often encode redundant information, the parameters of variational circuits evolve more freely due to limited entanglements in the circuits. Thus, a single scalar threshold may not be able to capture the true importance of a parameter component and may even lead to a degradation of performance by incorrectly deleting essential parameters. An adaptive procedure that prunes parameters based on the dynamics of gradient descent is more powerful and ultimately yields a superior set of sparse parameters. We introduce the ***QAdaPrune*** algorithm to perform pruning keeping in mind the aforementioned desiderdata. Algorithm 2 shows the overall algorithm and Figure 4.1 shows the algorithmic flow visually. We now discuss the key elements of the algorithm in more detail below.

Algorithm 2 The QAdaPrune Algorithm

procedure QAdaPrune $(\mathcal{D}, C(\boldsymbol{\theta}), w, T)$
  $\boldsymbol{\theta}_0 \leftarrow \text{INIT}(N)$ ▷ *N is the total parameter size*
  $\boldsymbol{\tau} \leftarrow \{1/N\}_{i=1}^{N}$
  $\mathcal{B} \leftarrow \nabla_\theta C(\boldsymbol{\theta}_0)$
  $\boldsymbol{s} = \{0\}_{i=1}^{N}$ ▷ *$\boldsymbol{s}$ is the saliency list*
  for $t = 0 \dots T$ **do**
    $\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} - \boldsymbol{\tau} \odot \sigma(\nabla_\theta C(\boldsymbol{\theta}))$
    $\mathcal{B} \leftarrow |\nabla_\theta C(\boldsymbol{\theta}_t) - \nabla_\theta C(\boldsymbol{\theta}_{t-1})|$
    if $t \bmod w = 0$ **then**
      $\boldsymbol{s} \leftarrow \text{BUILD\_SALIENCY}(\boldsymbol{\tau}, \mathcal{B})$
      $\mathcal{B} \leftarrow \nabla_\theta C(\boldsymbol{\theta}_t)$
    if $\boldsymbol{s} \neq 0$ **then**
      $\boldsymbol{\theta}_t^s \leftarrow \text{PRUNE\_PARAMS}(\boldsymbol{\theta}, \boldsymbol{s})$
    $\boldsymbol{\theta}_{t+1} \leftarrow \text{GRAD\_DESCENT}(\boldsymbol{\theta}_t)$
  return $\boldsymbol{\theta}_s^*$

Adaptive Threshold Detection: We introduce a component-wise adaptive threshold detection procedure that forms one of the key features of our algorithm. Before we elaborate on the specific update rule, we first take a deeper look at the optimization process. For a coordinate wise optimizer (i.e. an optimizer which updates each components independently) , we can realize the following update rule:

$$\theta_{t+1}^i = \theta_t^i - \eta \cdot g(\theta_t^i), \tag{4.1}$$

where θ_t^i is the ith component of the parameter vector $\boldsymbol{\theta}_t$, η is the learning rate and $g(\theta_t^i)$ is the gradient function applied to that component. Each parameter component has different outputs from the gradient component and hence for components θ^j and θ^i such that $j \neq i$, the optimization trajectory $\{\theta_0^i, \theta_1^i \dots \theta_T^i\}$ is different from optimization trajectory of $\{\theta_0^j, \theta_1^j \dots \theta_T^j\}$. This difference leads us to formulate the *independent parameter hypothesis*[1] that motivates this work: Each parameter in a VQC follows an independent trajectory and hence requires a threshold that is specific to the trajectory of that component.

[1] Here "independence" does not imply conditional independance.

In order to adaptively determine the threshold for each component we initialize $\boldsymbol{\tau} \in \mathbb{R}^L$ with $\tau_i \in \tau = 1/N$ with N being the parameter size. During the course of the optimization, we access the gradients $\nabla_\theta C(\boldsymbol{\theta})$ and update the threshold as

$$\boldsymbol{\tau'} = \boldsymbol{\tau} - \boldsymbol{\tau} \odot \sigma(\nabla_\theta C(\theta)) \tag{4.2}$$

where $\odot$ is element-wise multiplication, $\sigma(\nabla_\theta C(\boldsymbol{\theta}))$ is the softmax function given by

$$\sigma(\nabla_\theta C(\boldsymbol{\theta})) = \frac{e^{\nabla^j_\theta C(\theta)}}{\sum_{j=L}^{i} e^{\nabla^j_\theta C(\theta)}}. \tag{4.3}$$

Equation 4.3 returns the *relative importance* of each parameter component given the current time step's gradient. The threshold is then accordingly adjusted in response to the discovered importance value. The higher the importance of the component, the lesser the chance it will be pruned. Conversely, if the parameter is low in importance, the change in threshold will be significant and hence at a future time step the component is more likely to be pruned.

Better Pruning Criteria: In the proposed algorithm we perform the actual pruning (i.e., building a list of indices that indicate the components having the least impact towards the objective function) at fixed intervals of the optimization procedure. The length of this interval is governed by a user supplied variable w.

We have earlier mentioned that most pruning algorithms use the magnitude of gradient as a decision variable for pruning parameter components i.e. if $|\partial^j_\theta C(\boldsymbol{\theta}_t)|$ denotes the magnitude of j^{th} component of the gradient and γ is a scalar threshold, then the j^{th} component is pruned if

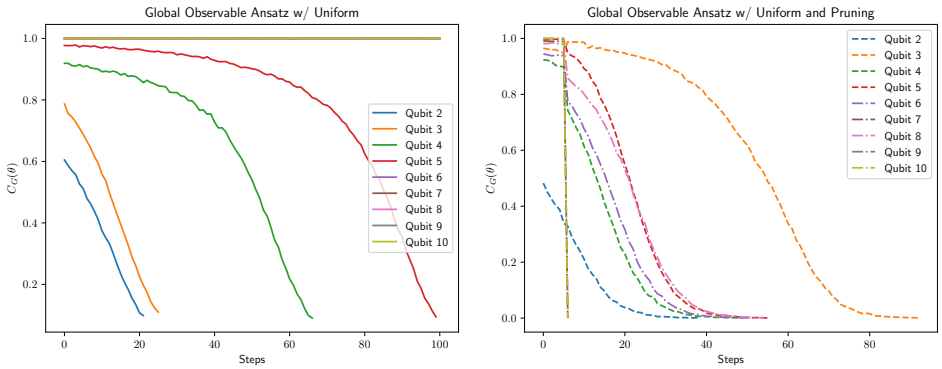
$$|\partial^j_\theta C(\boldsymbol{\theta}_t)| < \gamma.$$

In contrast, we choose to profile a different quantity - the absolute difference in magnitude of gradients at two successive time steps $|\nabla C(\boldsymbol{\theta}_{t+1}) - \nabla C(\boldsymbol{\theta}_t)|$. Thus, the j^{th} component is pruned if

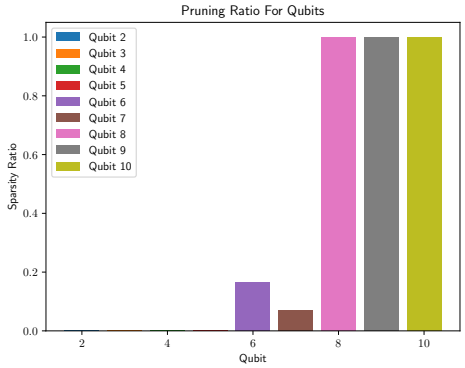
$$|\partial^j_\theta C(\boldsymbol{\theta}_{t+1}) - \partial^j_\theta C(\boldsymbol{\theta}_t)| < \tau^j$$

where we have defined $\partial^j_\theta C(\boldsymbol{\theta}) = \frac{\partial C(\boldsymbol{\theta})}{\partial \theta^j}$. It can be shown that

$$|\nabla C(\boldsymbol{\theta}_{t+1}) - \nabla C(\boldsymbol{\theta}_t)| \approx \nabla^2_\theta C(\boldsymbol{\theta}).$$



(a) VQC with Unpruned parameters *(b) VQC with Pruned Parameters*



(c) Pruning Ratios

Figure 4.2. *Experiments with VQC known to get stuck in barren plateaus. Figure 4.2a shows the training results for the qubits when no parameter pruning is employed. Figure 4.2b and 4.2c show the cases when pruning is employed and the resulting parameter sparsity. Pruning allows for training circuits upto 7 qubits (and with faster convergence), whereas the circuit gets stuck in barren plateaus for 6 or more qubits in unpruned case.*

Here $\nabla^2_\theta C(\boldsymbol{\theta})$ is the Hessian of the cost function and it has been shown[41,42] to be a much more useful quantity in pruning parameters. *The strength of our method is that we approximate this Hessian without explicitly computing it during the optimization procedure and thus automatically provide a more nuanced critera for pruning redundant parameters.*

4.2 Empirical Results

We now present empirical results by running QAdaPrune algorithm in different scenarios and applications.

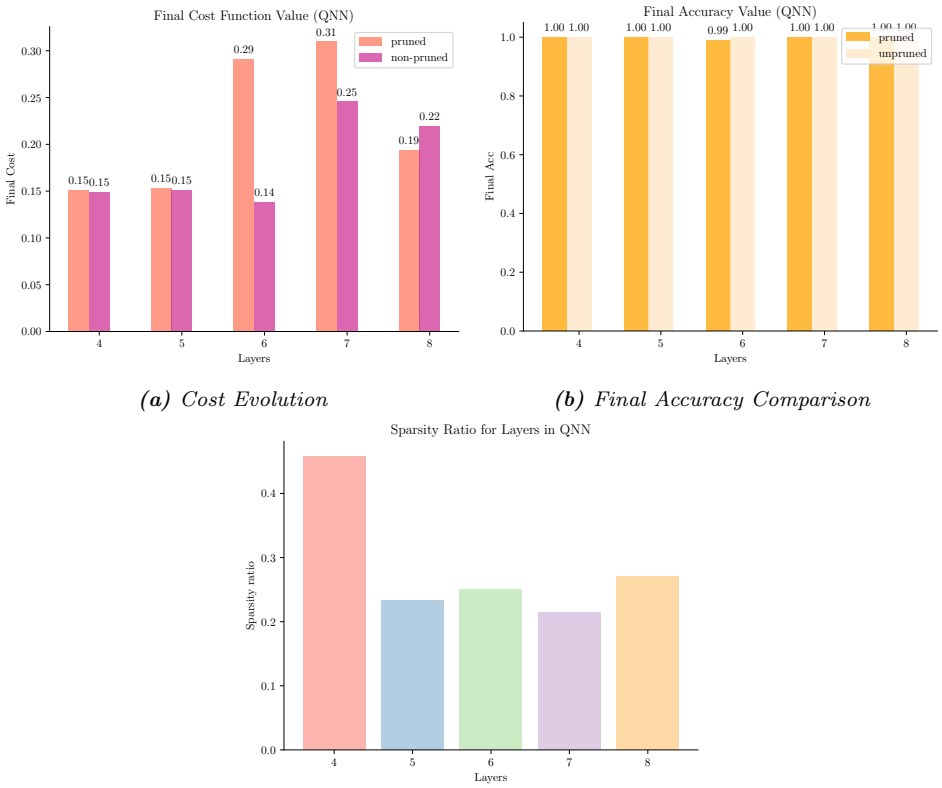
Exploring Trainability of VQCS In the first series of experiments, we aimed to

establish if pruning parameters can help in training deeper or more complex circuits. In order to accomplish this, we instantiated a hardware efficient ansatz with alternating single qubit gates R_X and R_Y. All qubits were then entangled in chained manner using Control-Z (CZ) gates. The numbers of layers was kept constant to 2 and the number of qubits was varied from 2 to 8. We initialized the parameters from a uniform distribution $\boldsymbol{\theta} \sim Unif(-\pi, \pi)$ and optimized the circuit such that $U(\boldsymbol{\theta})|0\rangle = |0\rangle$, i.e., the VQC was supposed to learn the identity function. All circuits were optimized using a gradient descent optimizer with learning rate 0.2. The results were obtained in the eigenbasis of the Pauli-Z global observable. A pruning window $w = 5$ was selected for all experiments.

The purpose of instantiating this circuit was that it is known to get stuck in barren plateaus as the number of qubits becomes higher when the parameters are initialized using the uniform distribution[12]. Our hypothesis was that pruning can help in alleviating this phenomenon to some extent. Figure 4.2 shows the results of our experiments. We can observe that when no parameter pruning is employed, the circuit gets stuck in barren plateaus for $n > 6$ where n is the number of qubits in circuit. However, when parameter pruning is employed, circuit gets into a barren plateau only after $n > 8$ and when the barren plateau does occur, the pruning algorithm automatically prunes all elements bringing the cost to zero. We can observe the resulting parameter sparsity in Figure 4.2c. Here, when the circuit is trainable, no parameter is pruned. When circuit starts to get stuck in barren plateaus, we are able to accomplish training by pruning $15 - 30\%$ parameters. These results show that an adaptive procedure like ours can help enhance trainability of VQCs.

Quantum Data Classification With Pruning

In the next series of experiments we consider the task of binary classification on a modified version of the Iris dataset[20] $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{100}$ where $\boldsymbol{x}_i \in \mathbb{R}^4$ is the data and labels are $y_i \in \{+1, -1\}$. To encode the classical data into a quantum state we employ angle embedding where for every j^{th} component of data vector i.e. $x_i^j \in \boldsymbol{x}_i$ we prepare



(a) Cost Evolution

(b) Final Accuracy Comparison

(c) Pruning Ratios

Figure 4.3. *Results of running the QAdaPrune algorithm on different layer QNNs with Iris Dataset. Figures 4.3a and 4.3b show the final cost and accuracy achieved by pruned and unpruned QNNs respectively. Figure 4.3c shows the percentage of parameters that were pruned by the QAdaPrune algorithm for different layers in the QNN.*

a rotation X gate with x_i^j being the fixed angle rotation. Thus, for a single qubit $K(x_i^j) = R_X(x_i^j) = e^{-i\frac{x^j\sigma_x}{2}}$, where σ_x is the Pauli-X matrix and is the gate generating Hermitian for the rotation X gate. We define our cost function

$$C(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} ||f(|\psi(\boldsymbol{x}_i, \boldsymbol{\theta})\rangle, \hat{\boldsymbol{O}}) - y_i||_2$$

as the L2 distance between the expectation of the resulting quantum state $|\psi(\boldsymbol{x}_i, \boldsymbol{\theta})\rangle$ and the label y_i. In our experiments, we keep the number of qubits $n = 4$ due to the dimensionality of the input data points. We vary the layers from $4 \rightarrow 8$ and observe the performance of our algorithm. As before, we initialize our parameters from a uniform distribution over the interval $[-\pi, \pi]$. We use the RMSProp optimizer[35] with a step size of 0.1. Each experiment is run for a 100 optimization steps with a pruning

window $w = 5$ on a quantum device with 1000 shots. The results of our experiments are shown in Figure 4.3. From Figure 4.3b we can see that the pruned and unpruned QNNs achieve the same accuracy on the test set despite the pruned QNNs converging to slightly worse final minima (shown in Figure 4.3a). Figure 4.3c shows the overall sparsity in the parameters after training. We can see for small depth circuits nearly 50% parameters are pruned. For deeper circuits, our algorithm is still able to prune roughly 25% of the parameters. This implies that with a dynamic pruning algorithm like ours, we can scale the training of QNNs to a larger depth without the concern of barren plateaus since the *effective depth* of the circuit will be shallow.

Experiments with VQE We now focus our attention on the problem of estimating the ground state energy of a molecule using a VQC[6]. In our experiment, we consider estimating the ground state energy of a Hydrogen(H_2) molecule with a UCCSD ansatz. We initialized the parameters from a normal distribution $\mathcal{N}(0, \pi)$ and used a gradient descent optimizer to run the algorithm for 40 steps with a step size of 0.5. The goal of our experiment was to investigate if our pruning algorithm can find redundant parameters in low parameter case. Figure 4.4 shows the energy estimation diagrams for the pruned and unpruned ansatz with the resulting parameter sparsity.

We can observe that at step number 5, the trajectory changes with a slight increase in the energy estimate. This occurs when the algorithm discovers a redundant parameter and is pruned. However, the pruned ansatz is still able to achieve the energy estimate found by the unpruned ansatz. This demonstrates additional evidence to our claim that QAdaPrune is an effective pruning strategy that is automatically able to recognize and prune parameters with no significant impact on the performance of the VQC.

4.3 Conclusion and Future Work

In this work we introduced an adaptive, online algorithm called QAdaPrune . The advantage of having such an algorithm is that it leads to more intelligent pruning by placing no global prior threshold on the parameters. We have demonstrated the effectiveness of our algorithm on classification and VQE tasks.

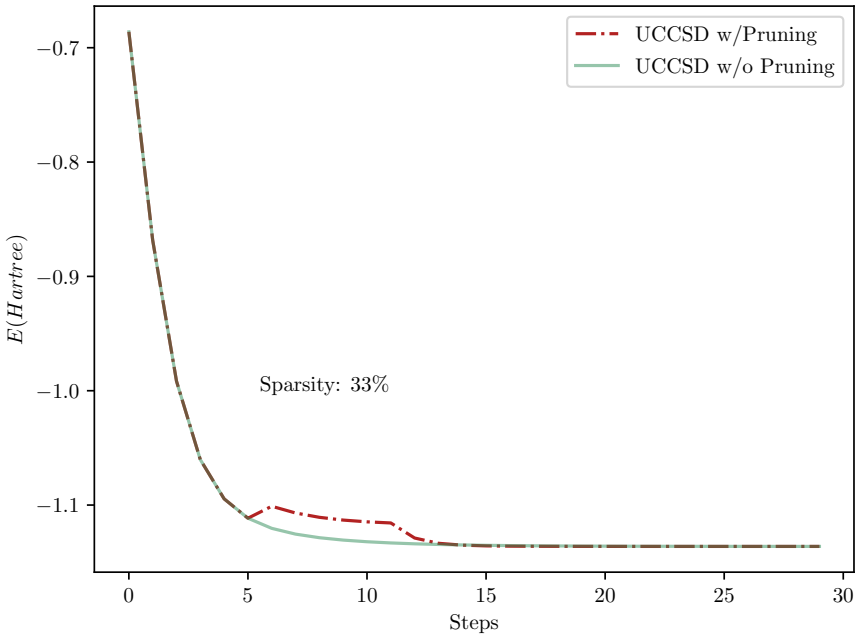


Figure 4.4. *Energy estimation of H_2 molecule with and without pruning. After the last pruning step we are able to prune atleast* 33% *parameters.*

In the future work we will explore how the choice of window size affects the pruning performance. We will further study if there are other ways to adaptively adjust threshold by drawing on techniques from classical machine learning. We believe that our procedure has the potential to enhance the trainability of variational circuits with minimal computational overhead and can be used to solve large problem instances of the aforementioned applications.

Chapter 5

FINDING EFFICIENT QUANTUM CIRCUITS VIA ARCHITECTURE SEARCH

In this chapter, we will discuss a new and emerging area in NISQ era quantum computing known as Quantum Architecture Search (QAS). We will introduce the key concepts and associated literature before introducing three algorithms developed during the course of this work for QAS in different application areas of quantum computing.

5.1 Architecture Search

In the typical setup of variational algorithm, we seek to find some optimal parameters $\boldsymbol{\theta}^*$ that correspond to the minimization of some cost function $C(\boldsymbol{\theta})$. The cost function is designed to model the underlying optimization problem in the given domain. In this setup it is assumed that there exists a "good" quantum circuit that has been hand designed to suit the given application. However,a manual design approach has several drawbacks. For instance, a manual design is prone to human errors and encourages ad-hoc design for different applications. Furthermore, a manual design may more likely be resource inefficient (in terms of gates/parameters).

Motivated by the shortcoming of manual design, Zoph *et al* proposed a *Neural Architecture Search*(NAS)[43] approach that automatically searches for the best *model* given input data and the application of interest. The QAS process is the quantum analog of the NAS structure and is shown in Figure 5.1. The key components common to both QAS and NAS are as follows:

- Search Space $\mathcal{S}$: The space encompassing all possible operations that are allowed in a possible neural network/PQC architecture.

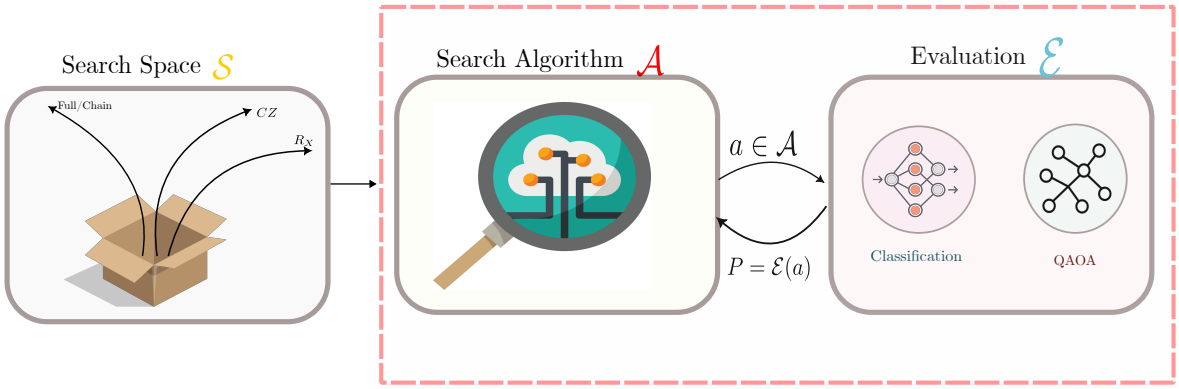


Figure 5.1. *A schematic representation of a QAS procedure.*

- Search Algorithm $\mathcal{A}$: The algorithm that is responsible for generating candidate architectures to be passed to the evaluator. It returns the optimal architecture given that performs best on the given application given the input data and satisfies any constraint that is provided at the time of search.

- Evaluator $\mathcal{E}$: The component that actually instantiates the generated architecture and returns the metric to signify the quality of the input architecture to the search algorithm.

In a NAS procedure, the search space includes all possible operations permitted in a neural network model and the search algorithm typically involves reinforcement learning methods to search for candidate architectures. The evaluation is carried out by running the generated neural network. Existing QAS methods also follow this similar setup, with the specific search space being different than the NAS search space. Different from these methods, we pursue the development of QAS procedure that meet the following goals:

- The search algorithm must return candidate *motifs* i.e. patterns of quantum circuits that perform well even when evaluated on significantly different data. These motifs are intended to be deployed in a layered architecture by repetition.

- The search algorithm must be able to model resource usage constraints. In other words, we want to integrate any resource constraints directly in search.

The goals of our current work differentiate us from other QAS methods and have perfomed favorably against hand designed circuits in the different areas of quantum algorithms considered in the scope of this work.

5.2 Related Work on Quantum Architecture Search

As mentioned earlier, there have been several methods for QAS that have been proposed in the literature.

Reinforcement Learning Based Methods: Fosel *et al*[44] consider the problem of optimizing the design of quantum circuits by first proposing inefficient circuits and then training a DNN to optimize the circuit given a desired circuit metric (e.g. number of gates). Ostaszewski *et al*[45] also propose to use deep reinforcement learning (RL) to obtain a good circuit for solving VQE[6] problem. A more general RL-based approach is considered in[46,47]. In the former work, the authors propose to realize a state preparation circuit given access to basic quantum gates and a target state. They leverage well known RL algorithms (PPO, A2C) for training the agent. The latter work also focuses on the same problem but assumes noise to be present in the realized circuit. They propose to use RL algorithms to derive gate sequences that can realize the target state in presence of noise. A different direction is taken by Zhang *et al.*[48] where they propose the quantum analog of the DARTS[49] work that aims to jointly model a possible gate operation and the weight associated with them in a single end-to-end differentiable framework. The problem is cast as a bi-level optimization problem where the parameters of the circuit are learned by gradient descent and the weights associated with the choice of the gate are optimized using RL policy gradient algorithms.

Non-Reinforcement Learning Based Methods: RL based methods have seen some success in specific problems. However, these methods often tend to take a long time to generate good architecture due to the sample inefficient nature of these methods. To get around the long time of search, different lines of inquires have emerged.

Duong *et al.*[50] propose to use Bayesian Optimization (BO) methods by treating a quantum circuit as a directed acyclic graph (DAG) and generating a QNN for a given data and cost function. BO based methods have the advantage that black box functions (e.g. performance indicator of a QNN) can be evaluated cheaply. However, the method is not scalable since BO becomes hard to evaluate as the number of dimensions of the associated variable grows. A different strategy is considered by Du *et al*[10] in which a quantum "supercircuit" is used to generate candidate quantum circuits that satisfy a given metric. The parameters are shared amongst all child circuits. The drawback with pure quantum strategy is that they are also non scalable to larger qubit sizes.

Evolutionary Search: Evolutionary search methods aim to generate the best circuit by modeling the search problem as a tournament selection. These algorithms typically generate random population of circuits/models and then perform crossover and mutation operations on those circuits to modify the population. The resulting best architectures are returned at the termination of the algorithm. Some work has been performed in this direction in[51,52] where the key task is to generate candidate circuits for VQE solver that can generalize to a variety of molecules. Different from classical machine learning where evolutionary search has been combined with neural methods[53] and has yielded non-trivial architectures at a fraction of computational cost of RL based methods, these methods remain largely unexplored in current QAS methods.

The three algorithms proposed in this work are instances of evolutionary search and a simplified RL based framework to tackle QAS based on the *structure* of the problem. We present the algorithm and results for max-cut QAOA and QNN based image reconstruction.

5.3 QAS for Max-cut QAOA Circuits

In this section, we describe the QAS procedure developed for combinatorial problems focussing specifically on the max-cut problem in graphs. We will first describe the

key concepts that are necessary to understand the algorithm and then present the algorithm and numerical results.

5.3.1 QAOA and Graph Max-Cut Problem

For a given simple undirected graph $G = (\mathcal{V}, \mathcal{E})$, the graph maxcut problem aims to find a maximum "cut set", i.e., a partition of nodes into two disjoint parts to maximize the number (or the total weight in case the graph is weighted) of edges that span both parts. The cost function for max cut is given as[2]:

$$C_{MC}(\boldsymbol{z}) = \frac{1}{2} \sum_{(u,v)\in\mathcal{E}} (1 - z_u z_v), \tag{5.1}$$

where $\boldsymbol{z}_i \in \{-1, +1\}$ is an indicator variable for node i that corresponds to the set membership the given node. In the QAOA setup, we start with an initial state $|s\rangle = |+\rangle^{\otimes n}$ where $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. A p layer alternating ansatz is then applied to the initial input state:

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = e^{-i\beta_p B} e^{-i\gamma_p C} \dots e^{-i\beta_1 B} e^{-i\gamma_1 C} |s\rangle. \tag{5.2}$$

Here, $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^p$ are parameters of the cost operator C and the mixer operator B, respectively. The cost function is measured by computing $\langle \boldsymbol{\gamma}, \boldsymbol{\beta} | C(\boldsymbol{z}) | \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle$. In QAOA problems, the structure of the cost operator C is generally guided by the problem we are interested in optimizing, but the structure of mixer operator is an open design problem. A measure of performance of a QAOA circuit is given by the approximation ratio:

$$r = \frac{\langle C_{quantum} \rangle}{C_{classical}} \tag{5.3}$$

This ratio measures the performance of a the QAOA algorithm against a classical solver on the same input graph. In the current NISQ regime, we want $r \in (0.5, 1.0]$.

5.3.2 Multi-Armed Bandits

In reinforcement learning, multi armed bandits are a class of algorithms that model a slot machine with k arms. Each arm is associated with a reward and the goal is to

Quantity	Defintion
$R(a)$	Reward for taking an action a
$Q(a)$	Estimated Q-value of action a
$Q^*(a)$	True Q-value of action a
ε	The exploration probability

Table 5.1. *Key quantities (and their notation) associated with multi armed bandits*

find out which arm gives the most reward over time. Contrary to other RL algorithms that model a *stateful* random process endowed with the Markov property, bandits are stateless algorithms. In other words, each action (pull of an arm) is independent of all other pulls. This property of multi armed bandits make them an attractive choice for QAS problems when the search space $\mathcal{S}$ consists of a small amount of gates and there exist a finite number of combinations for a possible architecture.

The key quantities to understand the mathematical formulation of bandits are presented in Table 5.1. In the bandit setup, $Q^*(a)$ is generally unknown. The goal then is to find an action that satisfies:

$$a^* = \arg\max_{a \in \mathcal{A}} Q(a) \tag{5.4}$$

There are two steps in a single run of a bandit algorithm. The first step is to choose a likely action from the given set of K-arms. During training, the bandit is encouraged to *exploit* it's knowledge (from past $Q(a)$ values) to select the arm with the best Q-value. With a finite probability ε, it is also encouraged to *explore* less likely action by sampling an action randomly. Experiments have shown that exploration can yield a higher overall reward as opposed to pure exploitation. The second step is to update the Q-value of the action based on the reward obtained by the pulling the arm. The update rule for $Q(a)$ is given as:

$$Q^{t+1}(a) = Q^t(a) + \alpha[R^t(a) - Q^t(a)] \tag{5.5}$$

The quantity $R^t(a) - Q^t(a)$ is the residual error in the Q-value estimation and α is the step size. If $\alpha \in (0, 1]$, then the reward corresponds to an exponential moving average

of previous time step rewards. If $\alpha = \frac{1}{n}$ (as in our work), the rewards correspond to the sample average over all possible actions. We direct the interested reader to the work by Sutton *et al*[54] for a deeper reading on multi-armed bandits.

5.3.3 QArchSearch and Results

The algorithm for searching the best QAOA mixer circuit is described in Algorithm 3.

Algorithm 3 `QArchSearch` for QAOA Mixer

Require: $\boldsymbol{\theta}$: parameters of quantum circuit, $\mathcal{A}_R$: Gate alphabet, p_{max}: depth of QAOA ansatz, K_{max}: maximum number of possible gate combinations G: input graph

$U_B^{best} \leftarrow \phi$
for $p : 1 \ldots p_{max}$ **do**
    energies $\leftarrow \{\}$
    for $k : 1 \ldots K_{max}$ **do**
        gate_comb $\leftarrow$ GET_COMBINATIONS($\mathcal{A}_R$, k)
        $\hat{U}_B \leftarrow$ BUILD_MIXER_CKT(G, gate_comb)
        $U_{QAOA}(\boldsymbol{\theta}) \leftarrow$ BUILD_QAOA_CKT($\hat{U}_B, p$)
        $\langle C \rangle \leftarrow$ SIMULATE_QAOA($G, U_{QAOA}(\boldsymbol{\theta})$)
        energies $\leftarrow$ APPEND(energies, $\langle C \rangle$)
    $U_B^{best} \leftarrow$ SELECT_BEST(energies, U_B^{best})
return
U_B^{best}, $\langle C_{best} \rangle$

In this algorithm we assume a finite gate alphabet $\mathcal{A}_R$ and k possible gate combination for a mixer circuit. Our aim is to find the mixer circuit that yields the highest cut energy when $p = 1 \ldots p_{max}$. For each p, we wish to evaluate a k gate combination from K_{max} possible gate combinations. We divide these goals into two distinct steps - global and local bandit steps.

The local bandit step is an instance of a *dynamic* ensemble of bandits. The dynamic part of the ensemble arises from the dynamic instantiation of the arms of the local bandit. These arms correspond to k^{th} order combination from the given gate alphabet $\mathcal{A}_R$. Once all local bandits have returned their best action arm the information about the best combination for a particular p is aggregated through a global aggregation step.

The global step is responsible for selecting the best *bandit* from amongst all bandits for any given p. This two step process allows the algorithm to explore a vast combination space efficiently and find non-intuitive designs that outperform hand designed ones.

5.3.3.1 Results of QArchSearch

We ran the search procedure with the gate alphabet $\mathcal{A}_R = \{R_X, R_Y, R_Z, H, P\}$ gates. Here R_σ is a rotation gate of the form $U(\theta) = e^{-i\sigma\theta/2}$ and σ is an element of the Pauli group. The other gates are Hadamard and Phase gates respectively. We evaluated candidate architectures for $p = 1 \ldots 4$ and $k = 1 \ldots 4$ on a dataset of 20, 10-node Erdos-Renyi (ER) graphs. The optimal mixer circuit discovered by the search procedure is shown in Figure 5.2.

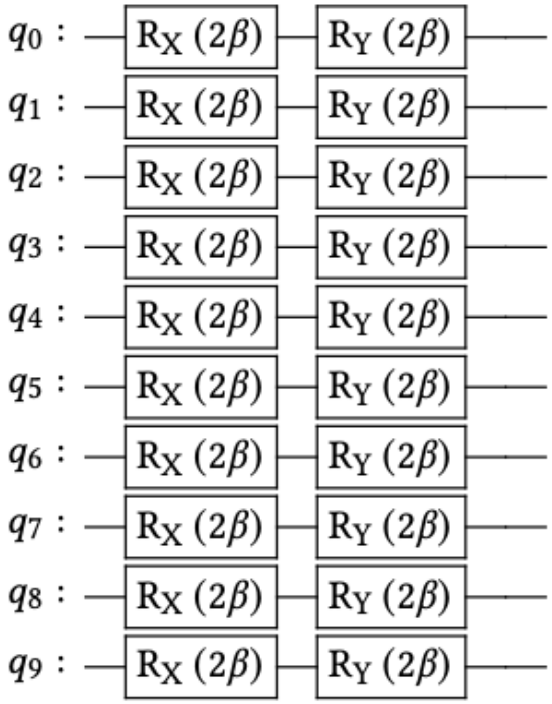


Figure 5.2. *The mixer circuit discovered by the QArchSearch algorithm.*

We evaluated the discovered mixer circuit on a collection of 20, 10-node 4-regular graphs. The results of that evaluation along with other best performing combinations are shown in Figure 5.3. We can observe that the discovered mixer circuit is able to perform well on unseen graph data instances as well. This lends evidence to our claim

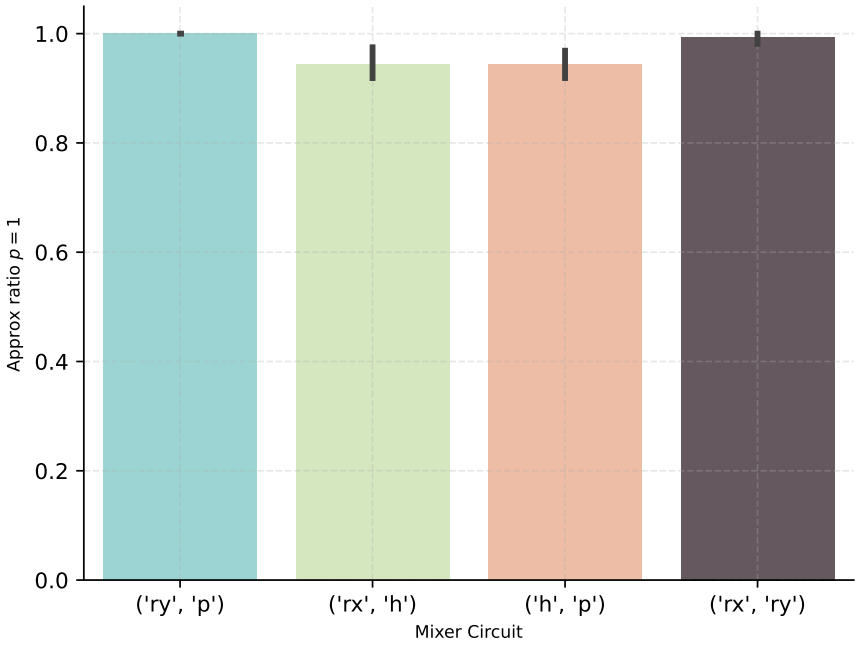


Figure 5.3. *Evaluation of mixer circuit performance on 4-regular graph dataset. The discovered mixer circuit performs well on unseen graph instances*

that the discovered mixer circuit is a design motif rather than an artifact of search on a specific dataset.

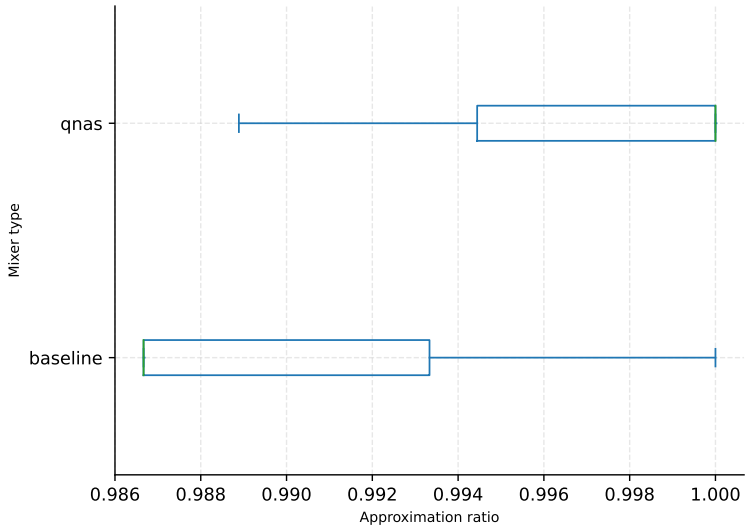


Figure 5.4. *Evaluation of mixer circuit for $p = 1, 2, 3$ for ER graphs.*

We further benchmarked the performance of the discovered mixer circuit against the

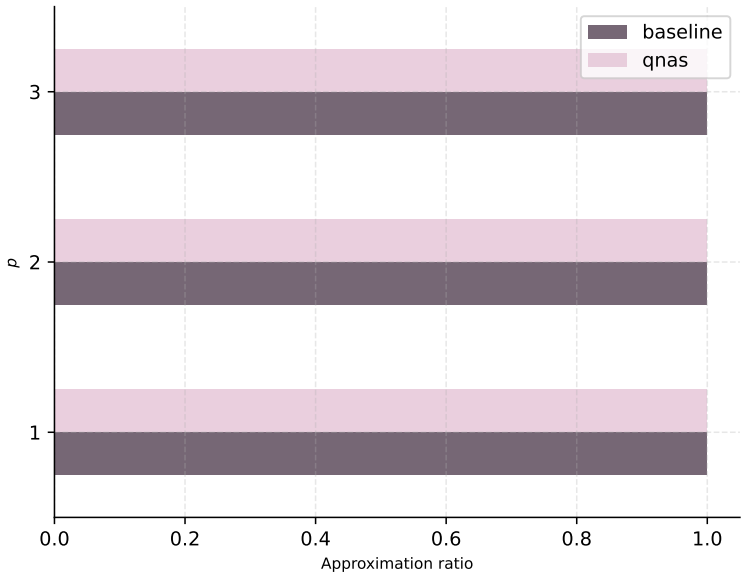


Figure 5.5. *Evaluation of mixer circuit for $p = 1, 2, 3$ for random 4-regular graphs.*

baseline version for $p = 1, 2, 3$ on the dataset of ER graphs and 4-regular graphs. The results are shown in Figure 5.4 and Figure 5.5. We can see that for ER graphs, the discovered mixer circuit outperforms the baseline circuit. In the case of random regular graphs, it performs equal to the baseline version. These results further show the quality of our solution.

5.4 QAS for Quantum Machine Learning

In this section we will present the quantum data compression problem and investigate if we can search for efficient quantum autoencoder architectures using QAS techniques. We will then present two algorithms that result from our investigation and show the efficacy of our results in two different image datasets.

5.4.1 Quantum Autoencoders & Quantum Data Compression

We begin by defining the quantum data compression problem below:

Definition 2. ***Quantum Data Compression****: Let $|\psi\rangle \in \mathbb{C}^m$ be a quantum state in a given Hilbert space $\mathcal{H}$. The problem of quantum data compression is to find a state*

$|\phi\rangle \in \mathbb{C}^n$ where $n < m$ and $|\phi\rangle \in \mathcal{H}$ such that the fidelity $\mathcal{F}(|\psi\rangle, |\hat{\psi}\rangle)$ is maximized. Here, $|\hat{\psi}\rangle$ is the reconstructed state from $|\psi\rangle$.

The main idea in quantum data compression problem is to accept an input *quantum* state and produced an output quantum state that contains the same information but in a lower dimensional space. In classical deep learning, such a transformation is achieved by an AutoEncoder (AE) model that acts on an input data $\boldsymbol{x} \in \mathbb{R}^m$ and produces a latent vector $\boldsymbol{z} \in \mathbb{R}^n$ via the transformation:

$$\boldsymbol{z} = Enc(\boldsymbol{x}, \boldsymbol{\theta}) \tag{5.6}$$

Where $\boldsymbol{\theta}$ are the parameter of the encoder. The reconstruction $\boldsymbol{x}$ is obtained by applying a decoder model to the latent vector. The model's parameters are trained by minimizing the reconstruction loss between the input and output.

A straightforward approach to solving the quantum data compression problem can be to train a variational circuit that mimics the behavior of the classical autoencoder model. However, translating an AE model into a Quantum AutoEncoder (QAE) is not a straightforward problem owing to the properties of quantum mechanics and the linear nature of the transformations caused by a quantum circuit. We now briefly present a method proposed in literature to realize a possible QAE model.

Quantum Autoencoder: A version of QAE model is proposed by Romero *et al*[9]. In this version the input quantum state is interpreted to be a bi-partite state i.e. $|\psi_{in}\rangle \in \mathcal{H}^A \otimes \mathcal{H}^B$. The subsystem A is called the latent system and the subsystem B is called the *trash* system. Essentially, we want to project the data contained in the original bi-partite state into a vector spanning only $\mathcal{H}^A$. A diagram representing the transformation is shown in Figure 5.6

The simplest cost function to train the parameters of the unitary $U_{AB}(\boldsymbol{\theta})$ will be to measure:

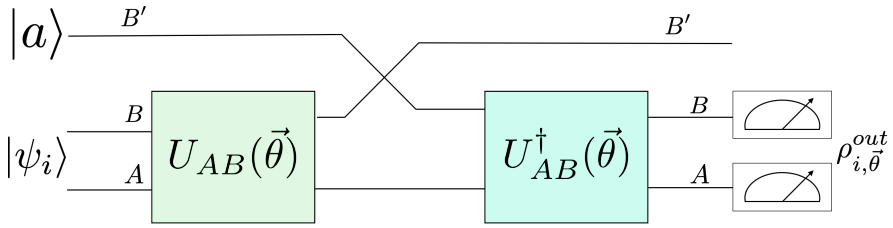


Figure 5.6. *A diagrammatic representation of a QAE model presented by Romero* et al.

$$C_1(\boldsymbol{\theta}) = \sum_i p_i \mathcal{F}(|\psi_i\rangle, \rho^{out}_{i,\boldsymbol{\theta}}) \tag{5.7}$$

Where $\{p_i, |\psi_i\rangle\}$ are ensembles of pure input states. The fidelity between two states is defined as $\mathcal{F}(\rho_1, \rho_2) = Tr[\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}]^2$. Even though this function is simple in theory, to achieve a good set of parameters, multiple copies of input states will be needed. This approach is then not very scalable.

To get around this limitation, the authors propose to prepare a known quantum state (a.k.a reference state) $|a\rangle$ in a subsystem B' and reduce the learning task in Equation 5.7 to performing a SWAP test on the trash subsystem only. This reduction simplifies the design of the autoencoder since the decoder unitary can simply be the inverse of the encoder unitary. The second cost function is given by:

$$C_2(\boldsymbol{\theta}) = \sum_i p_i \mathcal{F}(Tr_A[U_{AB}(\boldsymbol{\theta})[\psi_{AB}]U^{\dagger}_{AB}(\boldsymbol{\theta})], |a'_B\rangle) \tag{5.8}$$

Where $[\psi_{AB}] = |\psi_{AB}\rangle\langle\psi_{AB}|$. The quantity $Tr_A[U_{AB}(\boldsymbol{\theta})[\psi_{AB}]U^{\dagger}_{AB}]$ is the "traced-out" quantum state in the B sub-system after applying the parameterized unitary U_{AB}. The SWAP test measures the fidelity between the trash state and the reference state and allows for a simplified QAE model. C_2 and C_1 are not equivalent (the authors show that $C_1 \leq C_2$). However, for practical purposes C_2 can reliably compress quantum data.

In this work we are not interested in redefining the QAE model. Instead, we focus on searching the most *efficient* encoder circuit given some input data. The motivation for developing QAS techniques for this case is two fold. First, Romero *et al.* do not

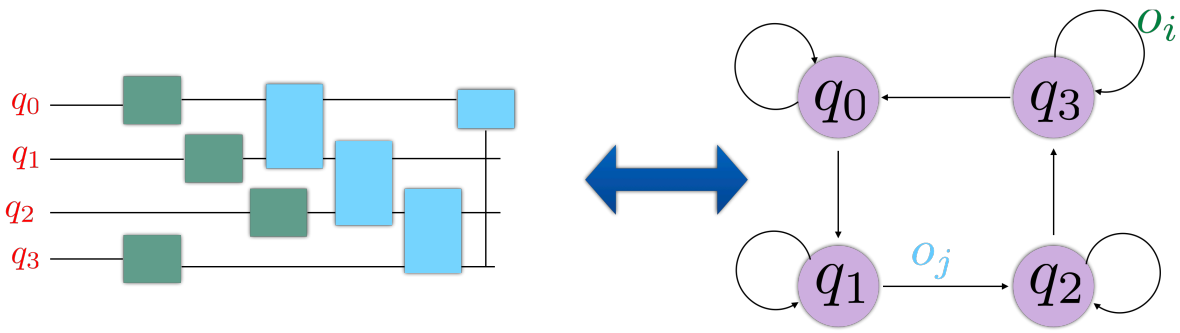


Figure 5.7. *A directed graph representation of a quantum circuit. The self loop o_is correspond to single qubit operation and o_j correspond to two qubit operations.*

provide a good guideline for designing the encoding unitary and hence the design is forced to be ad-hoc and inefficient. Second, we wish to create a minimal overhead for downstream classification/processing tasks.

Our goal is to create a search procedure that can search for *motifs* of quantum circuits that can serve as candidate encoding unitary. The discovered motif can then be repeated across many layers to improve performance. To achieve this goal, we create an intermediate representation of a quantum circuit as a directed graph. The graph consists of self loops and directed edges (See Figure 5.7). The self loops indicate sequential single qubit operations applied to that particular qubit. The directed edges between two qubits indicate one or more two-qubit operations. Typically, the two qubit operations are controlled entanglements and hence a directed edge between two nodes can be interpreted as a controlled entanglement operation where the control qubit is the source node of the edge. Armed with this intermediate representation, we introduce the two search algorithms.

5.4.2 Random Elastic Search (R.E.S)

One algorithm for searching for best encoding unitary circuit is described in Algorithm 4.

This algorithm combines the ideas of random search and incremental exploration. An important feature of this algorithm is that it has the ability to model *soft constraints* directly in the search process. These soft constraints can take the form of the depth, the

Algorithm 4 Random Elastic Search (R.E.S)

```
1: C: The soft constraint of the search
2: c ← 0
3: best_cell ← φ                                   ▷ Cell is an atomic unit of search
4: while c < C do
5:     if c = 0 then
6:         cells ← GEN_RANDOM_POP(POP_SZ)
7:         best_cell ← GET_BEST(cells)
8:     else
9:         cells ← GEN_RANDOM_POP(POP_SZ, best_cell) ▷ Uses previous best
           as seed for new search.
10:        best_cell ← GET_BEST(cells) ▷ Compares against current and previous
           best cells
11:    c ← c + 1
12: return best_cell
```

number of parameters or the number of single and multi qubit operations in the given motif. During initialization, the algorithm generates a random population of simple cells i.e. each qubit has one single qubit operation and one multi qubit operation. These cells are converted to their circuits and the best cell is selected to go on the next stage.

The second stage of the algorithm is the incremental expansion stage. Here, the best cell returned from the previous scan is used as a *baseline* and operations are explored by fixing operations that already exist in the cell. The generation and expansion stages alternate until the soft constraint is satisfied. Then a global best cell is returned at the end of the procedure. An advantage of this algorithm is that it can also be used as a subroutine in many evolutionary search algorithms.

5.4.3 Regularized Evolution with Learned Mutation (R.E.L.M)

RES algorithm is an extension of a random search method. In a way, it can be interpreted as a random search process where good initial points are found during the course of a search. The procedure also requires random search to be run after every expansion stage and can prove to be computationally intensive when the soft constraint

is very large. Motivated by these shortcomings and inspired by the success of evolutionary methods in NAS methods, we propose a more sophisticated search procedure described in Algorithm 5.

Algorithm 5 Quantum Architecture Search with Learned Mutation

Require: $\mathcal{C}$: Soft constraint of search, E: number of epochs, $\mathcal{P}$: Initial population size, $\mathcal{S}$: Sample size, $\boldsymbol{\phi}$: Parameters of mutation controller

$P_{init} \leftarrow$ GET_INITIAL_POPULATION$(\mathcal{P}, \mathcal{C})$ ▷ *Can use R.E.S if* $\mathcal{C} > 0$
$P \leftarrow P_{init}$
for $i : 1 \mapsto E$ **do**
  $S_{pop} \leftarrow$ SAMPLE$(\mathcal{S}, P)$
  $B, W \leftarrow$ GET_BEST_WORST(S_{pop})
  $C \leftarrow$ MUTATE$(\boldsymbol{\phi}, B)$ ▷ *Call to Mutation Controller*
  $f_C, r_i \leftarrow$ EVALUATE(C)
  $\boldsymbol{\phi}' \leftarrow$ UPDATE_CONTROLLER$(\boldsymbol{\phi}, r_i)$ ▷ *Use Policy gradient algorithm.*
  $P \leftarrow$PUSH_POP(W, C)
return B

Our proposed algorithm is called Regularized Evolution with Learned Mutation (RELM). There are two key features of our algorithm that make it quite novel in QAS family of algorithms. First, this algorithm leverages regularized evolution[55] for QAS and second, it takes inspiration from RENAS[56] algorithm to train a RL agent for *learning* a good mutation to existing circuit.

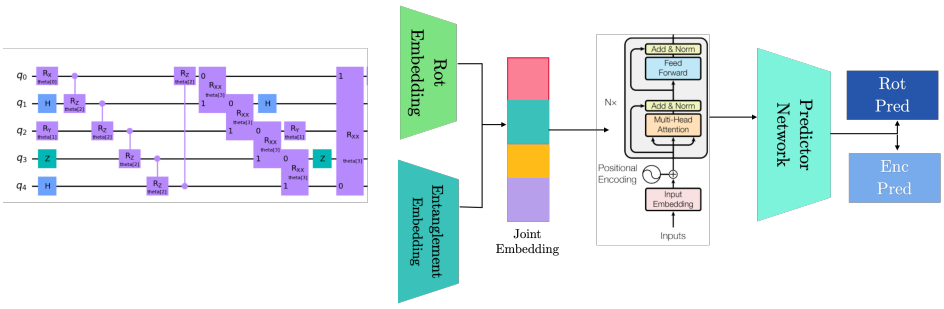


Figure 5.8. *Pipeline for circuit encoding in RELM procedure. A transformer model attends over embeddings and return the most likely mutation for the given input circuit.*

$$\mathrm{r}_i = \begin{cases} f_{parent} - f_{child} & \text{if } f_{parent} > f_{child} \\ tan(f_{child}\frac{\pi}{2}) & \text{otherwise} \end{cases} \tag{5.9}$$

At the start of the algorithm, we instantiate an initial population P_{init} using the `GET_INITIAL_POPULATION` method. This method can be a simple random search or a RES instance. If using the latter, then we can also encode the soft constraint directly in the initial population. Once we have the initial population we train the RL agent by repeatedly sampling from the initial population and finding the best and worst cells. The worst cell is removed from the population and the best cell undergoes the mutation process (described later). The mutated cell is evaluated and a reward obtained as per Equation 5.9. The controller's parameters $\boldsymbol{\phi}$ are updated using the Policy Gradient Loss[57] and the mutated cell is pushed into the population if it improves upon the performance of the parent.

In order to perform the mutation operation, one needs to translate the cell representation into a tensor representation that can be understood by a neural network. The encoding pipeline for such an operation is shown in Figure 5.8. The rotation and embedding gates first embedded into their own vector spaces. Then a joint embedding is created by concatenating the two embeddings. A Transformer[58] model then attends over this joint embedding and passes the result to a prediction module that predicts the rotation and entanglement mutation respectively. The specific operation is obtained by taking the index of the maximum value and using it in the search space.

5.4.4 Results on Image Datasets

We perform the task of finding the best unitary for the QAE by creating two synthetic image datasets. The first dataset is the Tetris dataset which consists of 500 images of Tetris blocks and are divided into four distinct categories. Each image is of the size 4×4. The second dataset is the Digits dataset which consists of 100 images of the numbers 0 and 1 in an image of size 8×4. The image datasets are shown in Figure 5.9. We perform an evaluation of the QAE resulting from the discovered unitary circuits by the two aforementioned search algorithms.

To evaluate the quality of circuits, we benchmark the circuits discovered by the algorithms against a baseline circuit shown in Figure 5.10. The discovered circuits are

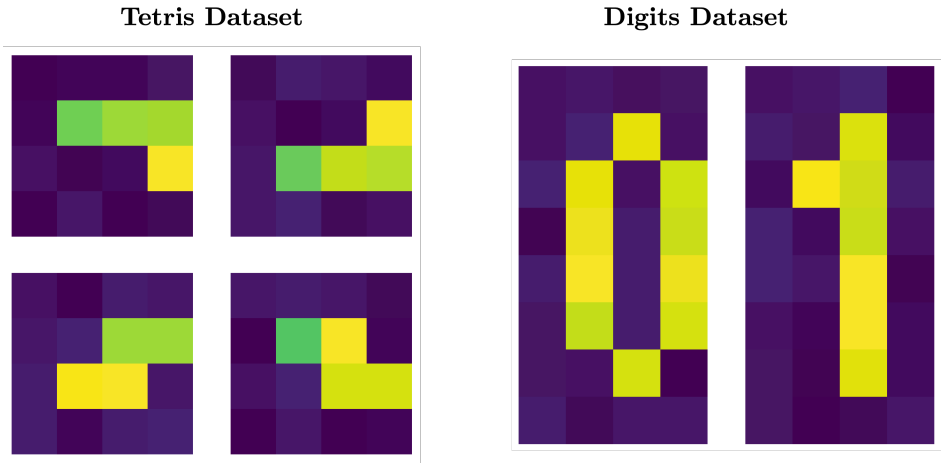


Figure 5.9. *The image datasets considered for quantum data compression in this work.*

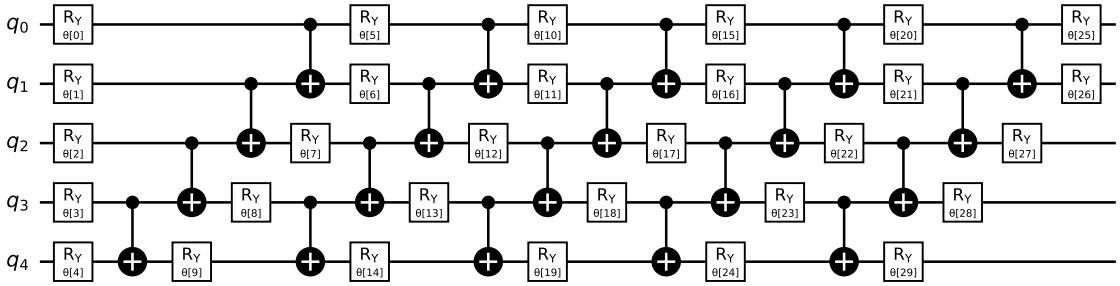


Figure 5.10. *Baseline encoder unitary for QAE experiments.*

profiled for their parameters and depth. In addition, we also measure the average test set fidelity and average cosine distance (on the test set). The cosine distance is given as:

$$cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T \boldsymbol{y}}{||\boldsymbol{x}||\ ||\boldsymbol{y}||}$$

5.4.4.1 Results on Digits Dataset

Table 5.2 shows the performance of Algorithms 4 and Algorithm 5 on the digits dataset. We evaluate the test set fidelity and cosine distance of a set of 50 images of randomly chosen 0s and 1s. Our goal is to evaluate the performance of the *motif* in the discovered circuit when compared to a hand designed baseline circuit.

Property	Baseline	RES	RELM + RS	RELM + RES
Num Parameters	30	14	10	24
Depth	23	10	6	14
Avg Fidelity(%)	86.19	78.00	80.60	87.18
Avg Cosine Distance(%)	89.90	77.37	79.36	82.52

Table 5.2. *Summary of various properties of QAE resulting from different search algorithm.*

The RES algorithm finds a parameter and depth efficient circuit that is able to have a decent fidelity when compared with the baseline. The key point to note is that the circuit is able to achieve a competitive fidelity even with less number of parameters. This shows that the discovered motif is indeed a good candidate design and performance can be improved by stacking more layers of this design. We then benchmark the performance of RELM by first running it with a random search initialization and then with RES initialization. In the first case, we see that the algorithm finds an even more efficient circuit and improves the performance in terms of test set fidelity over RES algorithm. In the second case, we see that the discovered circuit outperforms all previously discovered circuits including the baseline in terms of the test set fidelity. From this case we can see that the combination of our proposed algorithms is indeed effective in yielding high performance and efficient circuit designs.

5.4.4.2 Results on Tetris Dataset

We now analyze the performance of our algorithm on the Tetris dataset. We create a test set by randomly choosing 250 images from the original 500 images. Table 5.3 shows a summary of the results obtained by our proposed algorithm.

We see that all discovered circuits are efficient in terms of the parameters and the ansatz depth. Surprisingly, RES algorithm significantly outperforms the baseline test fidelity and achieves a competitive cosine distance on the test set. In the case of RELM with random search initialization, we see that the performance drops below the baseline quite significantly. However, this drop can be attributed to the shortcoming of the random search process rather than RELM since we see that when RES is used as

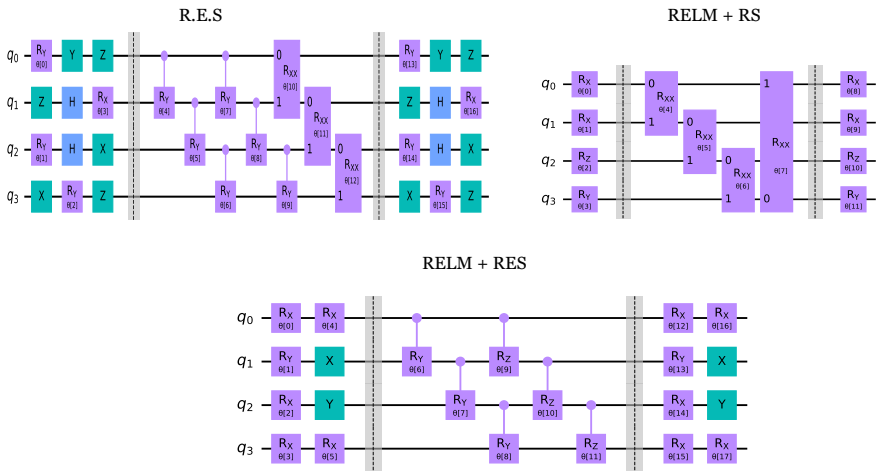


Figure 5.11. *Circuit structures returned by the two algorithms in three different configurations on the tetris dataset.*

Property	Baseline	RES	RELM + RS	RELM + RES
Num Parameters	24	17	12	18
Ansatz Depth	17	13	6	9
Test Fidelity(%)	71.30	93.58	63.54	85.30
Avg Cosine Distance (%)	92.02	90.16	77.78	87.50

Table 5.3. *Results on the profiled metrics against baseline on the Tetris Dataset. The fidelity and cosine distance are averaged over the test set.*

initialization method, the resulting circuit outperforms the baseline in terms of average test set fidelity. Furthermore, it achieves competitive performance with RES algorithm with a lower ansatz depth. Figure 5.11 shows the ansatz designs yielded by the different algorithms on this dataset. We can see that the circuits are significantly different than the baseline motif presented in Figure 5.10.

Chapter 6

THESIS CONCLUSION

In this thesis, we have introduced many new methods for scaling variational quantum algorithms on NISQ devices. Our overarching goal has been to take inspiration from classical machine learning techniques and propose effective algorithms that can potentially help quantum algorithms in overcoming the limitations of NISQ devices and convincingly demonstrate a quantum advantage.

In our first work we explored the lesser known relationship between parameter initialization and it's effects on the occurrence of barren plateaus in quantum circuits. We empirically showed that initializing parameters from the Beta distribution causes the variance of the gradient $\nabla C(\boldsymbol{\theta})$ to decay less than the usual case (i.e. $\boldsymbol{\theta} \sim Unif(0, 2\pi)$) by an order of magnitude. Based on this observation we propose the BEINIT algorithm that can be used to train deeper circuits without getting into barren plateaus. The empirical observations also led us to formulate a theoretical basis and show that changing parameter initialization from the uniform distribution can lead to a change in the *expressiveness* of the circuit and therefore change the rate at which the circuit gets into a barren plateau. This is a significant contribution in a fundamental problem of quantum computing in the NISQ era and our hope is that this unknown connection can inspire future work for quantum computing.

Our next work focused on a different but related problem - can we perform black box optimization of quantum circuits without computing gradients? To answer this question we developed a LSTM based MetaOptimizer that can learn to optimize a quantum circuit without computing gradients. We then showed that our optimizer can optimize circuits 3 times faster than gradient based optimizers. Since gradient computation is

a major bottleneck in variational quantum algorithms, our work fulfills a major need in the quantum computing community by creating the first version of gradient free optimization. A related direction to this work was the QAdaprune algorithm which proposed an adaptive parameter pruning procedure for removing redundant parameters and reducing optimization overhead. The novel aspects of the algorithm were that it determined the threshold adaptively and pruned parameters by computing an approximation to the Hessian of the cost function in $O(n)$ time.

Finally, we focused on an important problem that can be of use to a vast majority of quantum computing application - can we design a search algorithm that yields efficient and useful quantum circuits for any given application? To this end, we developed three quantum architecture search algorithms for two different applications - QAOA and QML. We demonstrated that the circuits discovered by our algorithms are non-trivial, parameter efficient and often outperform hand designed circuits on the given task. To the best of our knowledge, our RELM algorithm is the first instance of a Transformer[58] model being used in a quantum architecture search algorithm. We hope that these algorithms spark future innovations in quantum architecture search algorithms and result in circuits that can convincingly demonstrate a quantum advantage.

The future of quantum computing is promising and we expect that the development of fault tolerant quantum computers will result in quantum algorithms demonstrating their true potential. In the meantime, we envision that the algorithms proposed in this work and many other related works will help overcome the limitations of NISQ era quantum devices and will result in potentially useful quantum applications in the near future.

Bibliography

[1] Cerezo, M., Sharma, K., Arrasmith, A. & Coles, P. J. Variational quantum state eigensolver. *arXiv preprint arXiv:2004.01372* (2020). URL https://arxiv.org/abs/2004.01372.

[2] Farhi, E., Goldstone, J. & Gutmann, S. A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem. *arXiv preprint arXiv:1412.6062* (2014). URL https://arxiv.org/abs/1412.6062.

[3] Farhi, E., Goldstone, J. & Gutmann, S. A quantum approximate optimization algorithm. Preprint at https://arxiv.org/abs/1411.4028 (2014).

[4] Ushijima-Mwesigwa, H. *et al.* Multilevel combinatorial optimization across quantum architectures. *ACM Transactions on Quantum Computing* **2**, 1–29 (2021).

[5] Liu, X. *et al.* Layer VQE: A variational approach for combinatorial optimization on noisy quantum computers. *IEEE Transactions on Quantum Engineering* **3**, 1–20 (2022).

[6] Peruzzo, A. *et al.* A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* **5**, 4213 (2014). URL https://www.nature.com/articles/ncomms5213.

[7] Herman, D. *et al.* A survey of quantum computing for finance. *arXiv preprint arXiv:2201.02773* (2022).

[8] Farhi, E. & Neven, H. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002* (2018). URL https://arxiv.org/abs/1802.06002.

[9] Romero, J., Olson, J. P. & Aspuru-Guzik, A. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology* **2**, 045001 (2017).

[10] Du, Y., Huang, T., You, S., Hsieh, M.-H. & Tao, D. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information* **8**, 62 (2022).

[11] McClean, J. R., Boixo, S., Smelyanskiy, V. N., Babbush, R. & Neven, H. Barren plateaus in quantum neural network training landscapes. *Nature communications* **9**, 4812 (2018). URL https://www.nature.com/articles/s41467-018-07090-4.

[12] Cerezo, M., Sone, A., Volkoff, T., Cincio, L. & Coles, P. J. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications* **12**, 1–12 (2021). URL https://www.nature.com/articles/s41467-021-21728-w.

[13] Marrero, C. O., Kieferová, M. & Wiebe, N. Entanglement induced barren plateaus. *arXiv preprint arXiv:2010.15968* (2020). URL https://arxiv.org/abs/2010.15968.

[14] Verdon, D. Unitary 2-designs, variational quantum eigensolvers, and barren plateaus. Online at https://qitheory.blogs.bristol.ac.uk/files/2019/02/barrenplateausblogpost-1xqcazi.pdf.

[15] Bernardo, J. M. & Smith, A. F. *Bayesian theory*, vol. 405 (John Wiley & Sons, 2009).

[16] Gelman, A., Carlin, J. B., Stern, H. S. & Rubin, D. B. *Bayesian data analysis* (Chapman and Hall/CRC, 1995).

[17] Jin, C., Ge, R., Netrapalli, P., Kakade, S. M. & Jordan, M. I. How to escape saddle points efficiently. In *International Conference on Machine Learning*, 1724–1732 (PMLR, 2017).

[18] Ge, R., Huang, F., Jin, C. & Yuan, Y. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, 797–842 (PMLR, 2015).

[19] Neelakantan, A. *et al.* Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807* (2015).

[20] Dua, D. & Graff, C. UCI machine learning repository (2017). URL http://archive.ics.uci.edu/ml.

[21] Bergholm, V. *et al.* Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* (2018). URL https://arxiv.org/abs/1811.04968.

[22] Nesterov, Y. E. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, vol. 269, 543–547 (1983).

[23] Rudolph, M. S. *et al.* Orqviz: Visualizing high-dimensional landscapes in variational quantum algorithms. *arXiv preprint arXiv:2111.04695* (2021).

[24] Andrychowicz, M. *et al.* Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems* **29** (2016).

[25] Li, K. & Malik, J. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441* (2017).

[26] Wilson, M. *et al.* Optimizing quantum heuristics with meta-learning. *Quantum Machine Intelligence* **3**, 1–14 (2021).

[27] Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* **24** (2011).

[28] Verdon, G. *et al.* Learning to learn with quantum neural networks via classical neural networks. *arXiv preprint arXiv:1907.05415* (2019). URL https://arxiv.org/abs/1907.05415.

[29] Wu, Y., Ren, M., Liao, R. & Grosse, R. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021* (2018).

[30] Mnih, V. *et al.* Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[31] Harrow, A. W., Hassidim, A. & Lloyd, S. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* **103**, 150502 (2009).

[32] Schuld, M., Bergholm, V., Gogolin, C., Izaac, J. & Killoran, N. Evaluating analytic gradients on quantum hardware. *Phys. Rev. A* **99**, 032331 (2019). URL https://link.aps.org/doi/10.1103/PhysRevA.99.032331.

[33] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[34] Robbins, H. & Monro, S. A stochastic approximation method. *The annals of mathematical statistics* 400–407 (1951).

[35] Tieleman, T., Hinton, G. *et al.* Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* **4**, 26–31 (2012).

[36] Arunachalam, S., Grilo, A. B. & Yuen, H. Quantum statistical query learning. *arXiv preprint arXiv:2002.08240* (2020).

[37] Rakyta, P. & Zimborás, Z. Efficient quantum gate decomposition via adaptive circuit compression. *arXiv preprint arXiv:2203.04426* (2022).

[38] Wang, H. *et al.* Qoc: quantum on-chip training with parameter shift and gradient pruning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 655–660 (2022).

[39] Hu, Z. *et al.* Quantum neural network compression. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 1–9 (2022).

[40] Sim, S., Romero, J., Gonthier, J. F. & Kunitsa, A. A. Adaptive pruning-based optimization of parameterized quantum circuits. *Quantum Science and Technology* **6**, 025019 (2021).

[41] LeCun, Y., Denker, J. & Solla, S. Optimal brain damage. *Advances in neural information processing systems* **2** (1989).

[42] Hassibi, B., Stork, D. G. & Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, 293–299 (IEEE, 1993).

[43] Zoph, B. & Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[44] Fösel, T., Niu, M. Y., Marquardt, F. & Li, L. Quantum circuit optimization with deep reinforcement learning. *arXiv preprint arXiv:2103.07585* (2021).

[45] Ostaszewski, M., Trenkwalder, L. M., Masarczyk, W., Scerri, E. & Dunjko, V. Reinforcement learning for optimization of variational quantum circuit architectures. *Advances in Neural Information Processing Systems* **34**, 18182–18194 (2021).

[46] Kuo, E.-J., Fang, Y.-L. L. & Chen, S. Y.-C. Quantum architecture search via deep reinforcement learning. *arXiv preprint arXiv:2104.07715* (2021).

[47] Ye, E. & Chen, S. Y.-C. Quantum architecture search via continual reinforcement learning. *arXiv preprint arXiv:2112.05779* (2021).

[48] Zhang, S.-X., Hsieh, C.-Y., Zhang, S. & Yao, H. Differentiable quantum architecture search. *arXiv preprint arXiv:2010.08561* (2020). URL https://arxiv.org/abs/2010.08561.

[49] Liu, H., Simonyan, K. & Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[50] Duong, T. *et al.* Quantum neural architecture search with quantum circuits metric and bayesian optimization. *arXiv preprint arXiv:2206.14115* (2022).

[51] Rattew, A. G., Hu, S., Pistoia, M., Chen, R. & Wood, S. A domain-agnostic, noise-resistant, hardware-efficient evolutionary variational quantum eigensolver. *arXiv preprint arXiv:1910.09694* (2019). URL https://arxiv.org/abs/1910.09694.

[52] Chivilikhin, D. *et al.* MoG-VQE: Multiobjective genetic variational quantum eigensolver. *arXiv preprint arXiv:2007.04424* (2020). URL https://arxiv.org/abs/2007.04424.

[53] Real, E. *et al.* Large-scale evolution of image classifiers. In *International conference on machine learning*, 2902–2911 (PMLR, 2017).

[54] Sutton, R. S. & Barto, A. G. *Reinforcement learning: An introduction* (MIT press, 2018).

[55] Real, E., Aggarwal, A., Huang, Y. & Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 4780–4789 (2019).

[56] Chen, Y. *et al.* Renas: Reinforced evolutionary neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 4787–4796 (2019).

[57] Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* **12** (1999).

[58] Vaswani, A. *et al.* Attention is all you need. *Advances in neural information processing systems* **30** (2017).

ProQuest Number: 30990423

INFORMATION TO ALL USERS
The quality and completeness of this reproduction is dependent on the quality
and completeness of the copy made available to ProQuest.

Distributed by ProQuest LLC (2024).
Copyright of the Dissertation is held by the Author unless otherwise noted.

This work may be used in accordance with the terms of the Creative Commons license
or other rights statement, as indicated in the copyright statement or in the metadata
associated with this work. Unless otherwise specified in the copyright statement
or the metadata, all rights are reserved by the copyright holder.

This work is protected against unauthorized copying under Title 17,
United States Code and other applicable copyright laws.

Microform Edition where available © ProQuest LLC. No reproduction or digitization
of the Microform Edition is authorized without permission of ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346 USA