Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

Viroshaan Uthayamoorthy

Quantum Machine Learning in Variational Quantum Algorithms

Master's thesis in Applied physics
Supervisor: Franz Georg Fuchs
Co-supervisor: Alexander Johannes Stasik, Jeroen Danon, Halvor Møll Nilsen

June 2022

NTNU
Norwegian University of Science and Technology

SINTEF

Viroshaan Uthayamoorthy

Quantum Machine Learning in Variational Quantum Algorithms

Master's thesis in Applied physics
Supervisor: Franz Georg Fuchs
Co-supervisor: Alexander Johannes Stasik, Jeroen Danon, Halvor Møll Nilsen
June 2022

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

NTNU
Norwegian University of
Science and Technology

Abstract

This master thesis introduces concepts of machine learning and quantum machine learning. Particular focus will be put on variational quantum algorithms (VQAs) and how these quantum circuits are trained. Known issues related to the trainability of these hybrid quantum-classical computation methods are also discussed. Limiting the scope to the variational quantum algorithm QAOA, two different parameter initialization heuristics were implemented and analyzed on the problem of MaxCut. The analysis considers unweighted 3-regular graphs and Erdős-Rényi graphs in particular. These heuristics are referred to as INTERP and Parameter Fixing and were introduced by Zhou et al. [1], and Lee et al. [2] respectively. Both heuristics provide ways to incrementally increase the depth of the QAOA circuit by finding local optima at the current depth. Numerical results of both methods indicate that the underlying mechanisms are similar for both heuristics and that INTERP is preferable because of its lower run-time.

Based on trends discovered using the INTERP heuristic, a similar procedure to the one presented by Alam et al. [3] was implemented where machine learning is used to learn the patterns in these optimal QAOA parameters. Using this approach, one can reduce the number of intermediary optimization runs needed to find good local optima at a target depth. The thesis shows that relatively few graph instances are needed to learn the optimal QAOA parameters using a feedforward neural network. This machine learning procedure was tested on weighted 3-regular graphs.

Lastly, the thesis considers a realistic implementation of the ESCAPE algorithm of Rivera-Dean et al.'s [4] using a shot-based simulator instead of an ideal one. In this implementation, a gradient-free optimizer was used instead of a gradient-based one to reduce the number of function evaluations performed on a quantum computer. The procedure was tested on weighted five and twelve-node graphs, with the number of successful escapes being higher than the original procedure. This increase in performance is believed to be caused by convergence difficulties with gradient-free optimizers and random perturbations into regions of better cost.

Simulations on the five-qubit noise model FakeManilla from IBMQ show that the number of successful and failed escapes is comparable, and thus the procedure is less reliable on NISQ hardware.

The thesis bases its results on extensive numerical simulations. Over 20 000 CPU hours on computer clusters were used to run simulations on 90 graph instances to gather data for the ESCAPE routine. At each depth of the QAOA routine for each graph instance, 100 randomized initializations were used to gauge the performance of the algorithm. The INTERP routine and the subsequent usage of a neural network to perform parameter predictions was tested for 200 graph instances.

Acknowledgements

I would like to express my sincerest thanks to my main supervisor Franz Fuchs for providing informative articles, constructive criticism, and feedback throughout the duration of the thesis. Additionally, I would like to extend my gratitude to Halvor Møll Nilsen, Alexander Johannes Stasik, and Jeroen Danon for providing invaluable input to the thesis. The simulations were performed on resources provided by Sigma2 - the National Infrastructure for High-Performance Computing and Data Storage in Norway.

Sammendrag

Denne masteroppgaven introduserer konsepter innen maskinlæring og kvantemaskinlæring. Spesiell fokus vil bli ilagt variasjonelle kvantealgorithmer hvor parametre i kvantekretser blir optimert. Hvordan disse kretsene trenes of kjente problemer relatert til disse hybride beregningsmodellene blir diskutert gjennom oppgaven. Skopet til oppgaven begrenses til en spesifikk kvantealgoritme kalt QAOA. I forhold til denne algoritmen kommer to heuristikker innen parameter initialisering til å bli testet og analysert på beregningsproblemet MaxCut. Analysen er utført på uvektete grafer med tre kanter pr. node og diverse Erdős-Rényi graf instanser. Heuristikkene heter INTERP og Parameter Fixing. Disse heuristikkene ble henholdsvis introdusert av Zhou m.fl. [1] and Lee m.fl. [2]. Begge heuristikkene viser til måter for å inkrementelt øke dybden av en QAOA krets ved å ta i bruk parameterverdier fra et foreløpig lokalt minimum. Numeriske reultater for begge metodene indikerer at den underliggende mekanismen for begge metodene ligner og at INTERP er den foretrukne metoden på grunn av dens relativt lave kjøretid.

En utvidelse av dette arbeidet ble også utført basert på en metode lignende Alam m.fl [3] hvor maskinlæring blir brukt for å lære trendene i optimale QAOA parametre. Denne tilnærmingen gjør at man kan redusere antallet optimeringssteg for å øke dybden på QAOA kretsen til en målsatt dybde. Denne prosedyren ble implementert og testet på vektede grafer med tre kanter pr. node. Resultatene viser at relativt få graf instanser er nødvendig for at et kunstig nevralt nettverk skal kunne lære de optimale QAOA parametrene.

Til slutt bygger masteroppgaven videre på Rivera-Dean m.fl.'s [4] ESCAPE algorithme ved å implementere algoritmen på en mer realistisk måte. Dette gjøres ved å se på en skudd-basert kvantedatamaskin. Med formålet om å redusere antallet kall til kvantedatamskinen blir en gradient-fri variant av algoritmen utviklet og testet. Denne prosedyren er testet på vektede grafer med 12 og 16 noder hvor hver node er tilkoblet tre andre noder. Resultatene viser at denne varaisjonen er mer effektiv enn funnene til Rivera-Dean m.fl. Det er grunn til å tro at den høye suksessraten kommer av konvergensproblemer i de gradient-fri optimeringsprosedyrene og at tilfeldige perturbasjoner gjør at ESCAPE prosedyren finner området i kostlandskapet som har gode løsninger.

Simuleringer på en støy-modell fra IBMQ viser at antallet vellykkede of defekte kjøringer av algoritmen er sammenlignbare, og dermed er algortimen lite pålitelig på realistiske kvantedatamaskiner.

Resultatene som blir diskutert i master oppgaven stammer fra omfattende numeriske simuleringer. Over 20 000 CPU timer ble brukt for å kjøre simuleringer på 90 graf instanser for å hente inn data for ESCAPE rutinen. Dataene som ble brukt for å trene opp et nevralt nettverk til å predikere parametrene ved å bruke trender fra INTERP rutinen baserte seg på 200 graf instanser.

Contents

List of Figures

List of Tables

1 Introduction

Recent years have seen great strides in large-scale quantum computing, and interest has grown in potentially practical near-term applications. Several algorithms have already been designed to utilize some of the exciting capabilities of quantum computers, such as Shor's algorithm [5] for prime factoring and the HHL algorithm [6] for solving linear systems of equations. However, these traditional algorithms require a high number of qubits and long decoherence times that far exceed the capabilities of the quantum hardware available today. Describing the limitations of current-day quantum hardware is the term Noisy Intermediate-Scale Quantum Computing (NISQ) [7], a term coined by John Preskill to describe an era of quantum computing where noise and decoherence cause unreliable operations. To address these issues, hybrid quantum-classical algorithms have emerged as the most promising candidates, using the available quantum resources while utilizing classical routines to process quantum information. Examples of such algorithms are the Variational Quantum Eigensolver (VQE) [8] and the Quantum Approximate Optimization Algorithm (QAOA) [9] which were introduced to solve quantum chemistry problems and classical combinatorial optimization problems respectively.

The hybrid algorithms VQE and QAOA are two particulars of a more comprehensive class of algorithms called Variational Quantum Algorithms (VQA). These algorithms consist of two parts; one performed on a quantum computer and another on a classical computer. The quantum part of these algorithms consists of preparing a quantum state using a quantum circuit. A quantum circuit is a sequence of quantum gates (unitary operators) that act on single or multiple qubits. In the case of VQAs, the quantum gates involve parameters that can be adjusted. An example of these types of gates is a single qubit rotation gate, where the rotation angle is the variational parameter. The set of gates that defines the circuit structure is called an ansatz. These circuit designs can have elements that incorporate parts of the problem, often referred to as problem-dependent ansätze, or be problem agnostic in their structure. The evolution of the quantum state through an ansatz creates a final prepared state from which measurements are performed. The measurement output of the quantum system (often in the form of bitstrings) is the input to a cost function to optimize. Optimizing the free variational parameters for the cost function is performed using a classical optimizer. These optimized parameters are then fed into the quantum circuit to create a closed feedback loop. If the neural network block is removed, the rightmost subfigure of figure 1 illustrates this feedback loop between the quantum computer and classical optimizer.

Bittel et al. [10] find that the classical optimization of these hybrid algorithms is NP-hard when using random initial points due to the training landscape having multiple local minima with suboptimal cost. Hence, gradient-based optimizers will generally converge to these sub-optimal solutions. It is found that the quality of local minima of VQA undergoes a phase transition [11]. Below the transition, the expected distribution of local minima values is far from the values of the global minima, which supports the NP-Hard results derived by Bittel et al. [10]. However, the paper finds that once the number of parameters reaches a certain threshold, all local minima values are distributed close to the global one, hence improving the

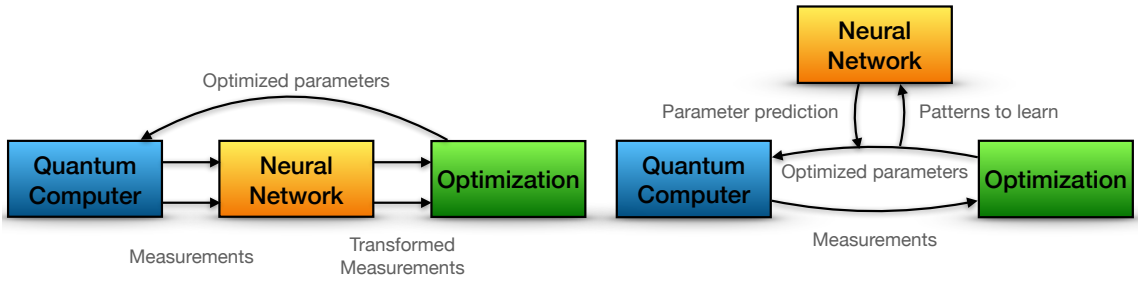


Figure 1: A schematic overview of the two ways classical neural networks will be included in the processing of quantum information in the master thesis. The rightmost version is elaborated on in section 6 while the leftmost version is presented in section 7.

quality of the local minima. The rub is that the number of parameters needed to achieve this transition is exponentially large in the problem size. This transition has also been found numerically in various problem-agnostic ansätze [12].

In contrast, it is widely recognized that heavily parametrized deep neural networks are significantly easier to train as the expected distribution of local minima values is concentrated around the values of the global minima [13, 14]. With these observations in mind, the motivation behind this master thesis is to add more parameters to VQAs by including a classical neural network to create a model that more reliably converges to optimal solutions. The leftmost subfigure of figure 1 illustrates this procedure. As the figure shows, the neural network transforms the measurements from the quantum computer before calculating the cost. The neural network introduces additional variables that are optimized using the classical optimizer. Numerical studies are conducted on problem-dependent ansätze to see if any significant improvements can be made by increasing the number of parameters through the neural network variables. This work extends on Rivera-Dean et al.'s [4] work on avoiding local minima in VQAs with neural networks.

The thesis is structured as follows. Section 2 presents a high-level overview of central concepts in machine learning, which ends in the introduction of feedforward neural networks. This machine learning method is heavily used throughout the thesis. From there, section 3 introduces quantum machine learning, including measurements, encoding procedures, and VQAs. These hybrid models are trained in section 4. Section 5 introduces the QAOA algorithm and applies it to the problem of MaxCut. This section presents two different parameter initialization heuristics, following a discussion of the results when applying them to MaxCut. Based on these findings, section 6 utilizes feedforward neural networks to create effective initialization strategies for the variational parameters for this particular VQA. The right subfigure of figure 1 illustrates this procedure. This figure shows that the neural network learns trends and patterns from the feedback loop to form efficient parameter predictions. Lastly, the main algorithm of the thesis, namely avoiding local minima in VQAs with neural networks, is introduced in section 7, schematically shown in the left subfigure. Along with a description of the procedure, numerical studies are conducted, and results are discussed. Note that the master thesis extends on the work of the specialization project, which contained sections 2 - 5.

The main work of the master thesis can be summarized as follows:

- Reproduction of the heuristic parameter initialization method INTERP of Zhou et al. [1] to find optimal QAOA parameters close to the global minimum of the MaxCut problem.
- Reproduction of a similar initialization heuristic; the parameter fixing heuristic as introduced by Lee et al. [2]
- A rigorous comparison between both heuristics finds similar trends in the QAOA parameters upon successful runs.
- Using the patterns found from the initialization heuristics, a feedforward neural network is used to learn these trends in optimal QAOA parameters. Based on these trends, parameter predictions are made to reduce intermediary optimization steps. This procedure is inspired by a similar procedure introduced by Alam et al. [3].
- Reproduction of the ESCAPE algorithm of Riveradean et al. [4] to escape potential local minima using a classical feedforward neural network.
- Worked on extending the method into a shot-based approach instead to see how it would work in a more realistic implementation of the algorithm that uses gradient-free methods. The motivation behind this extension is to limit the number of function evaluations used in the procedure.
- Tested the new version of the ESCAPE procedure on the IBMQ noise model FakeManilla to see how the algorithm would perform on real hardware.

2 Machine Learning

Machine learning is a field of study about creating algorithms where computers can learn without being explicitly programmed. Tom Mitchell proposed a widely quoted definition of machine learning in 1997. He defines it as the following:

> Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. (T. Mitchell p.2 [15])

To exemplify this definition, consider the following. The task T can be playing chess or creating an E-mail spam filter. The algorithm gains experience in the context of chess by playing other players or even other computers. In the case of the spam filter, gaining experience would be the algorithm sorting through E-mails labeled as spam or not spam, from which it learns to detect patterns. Performance measure P in the event of chess would be the win/loss rate, while in the spam filter case, the performance can be measured as the accuracy of the model's E-mail classifications as spam.

At the heart of machine learning lies data-driven decision-making and prediction, where models infer useful information from the implicit patterns in the data that give rise to the models. On an overarching level, machine learning is divided into three disciplines: reinforcement learning, supervised learning, and unsupervised learning. Only the latter two disciplines are relevant for this project assignment and thus are presented in the following chapters. Additionally, a high-level overview of the different elements of a machine learning algorithm will be presented in section 2.3. Lastly, feedforward neural networks are presented in section 2.5 in anticipation that this machine learning model is relevant for the master thesis.

2.1 Supervised Learning

Supervised learning algorithms build the predictive model from a data set containing training input $\boldsymbol{x}$ and corresponding training labels $\boldsymbol{y}$. Based on a limited training data set, the model infers a function that can predict the output of some previously unseen input. Schuld and Petruccione use the following definition in the book *Supervised Machine Learning with Quantum Computers*:

Definition (*Supervised learning task*): Given an input domain $\mathcal{X}$ and and output domain $\mathcal{Y}$, a training data set $\mathcal{D} = \{(\boldsymbol{x}^1, \boldsymbol{y}^1), \dots (\boldsymbol{x}^M, \boldsymbol{y}^M)\}$ of training pairs $(\boldsymbol{x}^m, \boldsymbol{y}^m) \in \mathcal{X} \times \mathcal{Y}$ drawn independently from the same underlying distribution $P(\boldsymbol{x}, \boldsymbol{y})$ (iid) with $m = 1, \dots, M$ of training inputs $\boldsymbol{x}^m$ and target outputs $\boldsymbol{y}^m$, as well as a new unclassified input $\tilde{\boldsymbol{x}} \in X$, predict the corresponding output $\tilde{\boldsymbol{y}} \in Y$ (Schuld and Petruccione p.25, [16]).

The input domain $\mathcal{X}$ is an N-dimensional vector space. The input-vector $\boldsymbol{x}$ are also called feature vectors since they represent information on selected features. A

prototypical example is to calculate the expected selling price of a house. In this case, the input would be an N-dimensional vector $\boldsymbol{x}$ where each element is an instance of a feature that is indicative of the pricing of a house (ex. number of rooms, location, size of the house, etc.). The output $\boldsymbol{y}$ is the corresponding price of that instance.

The dimension of the output-domain separates two types of supervised learning problems. The above example was an example of a *regression* problem. The goal of a regression algorithm is to predict the value of a continuous variable y, and thus the output space $\mathcal{Y}$ is the space of real numbers $\mathbb{R}$. In contrast, a *classification* problem is categorized as a problem in which the prediction $\boldsymbol{y}$ falls into a set of D discrete labels $\{l_1, \ldots, l_D\}$. The output-vector $\boldsymbol{y}$ is generally a real vector in $\mathbb{R}^D$ where each of the elements y_i is the probability of the model predicting class i. An example of a classification problem is classifying an image into three classes: dog, cat, or mouse. If an input results in predicted state $\boldsymbol{y} = (0.2, 0.7, 0.1)^T$, the model would classify that image as a cat.

2.2 Unsupervised Learning

During the training of an unsupervised learning algorithm, the algorithm is given a dataset that contains many features $\boldsymbol{x}$, but the data are not preassigned any labels. The goal of the unsupervised learning approach is to self-discover any useful properties of the structure of the dataset. These approaches are used to either make predictions on new input or more commonly used as a preprocessing of the data to reduce the complexity of the dataset. In general, one has access to an empirical distribution from M samples in a dataset, which can be expressed as

$$q(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} \delta(\boldsymbol{x} - \boldsymbol{x}^m) \tag{1}$$

As before, $\boldsymbol{x}^m$ refers to an instance of the input data, separated into its features. The goal of an unsupervised approach is to change the parameters $\boldsymbol{w}$ of a family of distributions $p(\boldsymbol{x}; \boldsymbol{w})$ so that it is similar to the empirical distribution $q(\boldsymbol{x})$ provided by the data [13]. In other words, these approaches modify p such that the probability of measuring data points from the data set is high.

A definition of these types of tasks as given by Schuld and Petruccione in the book *Machine Learning with Quantum Computers* is the following:

Definition (*Unsupervised learning task*): Given an input domain $\mathcal{X}$ and a data set of input samples $\mathcal{D} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^M\}$ with $\boldsymbol{x}^m \in \mathcal{X}$, drawn from a probability distribution $p(\boldsymbol{x})$, approximate the probability distribution $p(\boldsymbol{x})$ and draw a new sample $\boldsymbol{x}$ using the approximated distribution (Schuld and Petruccione, p.27 [17]).

Common unsupervised learning algorithms are clustering- and dimensionality reduction techniques. Clustering is an approach in which the algorithm groups the training examples into categories with similar features. Examples of these types

of algorithms are K-means clustering [18], hierarchical clustering [19], and probabilistic clustering [20]. Dimensionality reduction is a group of techniques in which the algorithm reduces the necessary number of features in the dataset while still preserving the underlying patterns. Common algorithms are principal component analysis (PCA) [21], singular value decomposition (SVD) [22], and autoencoders [23].

2.3 Elements of Machine learning

As indicated in the above two subsections, the task of the machine learning routine is to pick the best model that can reproduce the data from the data set. The best model f is chosen from a set of functions $\{F\}$ which can be a mapping from the input domain $\mathcal{X}$ to output domain $\mathcal{Y}$, or a distribution p over the input domain $\mathcal{X}$.

Choosing the optimal model is different for the supervised and unsupervised learning tasks. For supervised learning tasks, this process is known as empirical risk minimization. Since the data is labeled in these tasks, one can define the loss function $L(f(\boldsymbol{x}), \boldsymbol{y})$ which is a measure of the error between the model prediction $f(\boldsymbol{x})$ and the target $\boldsymbol{y}$ for said input. Risk is defined as the expected loss over the true distribution p over the input and output pairs $(\boldsymbol{x}, \boldsymbol{y})$,

$$R_{\text{true}}(f) = \mathop{\mathbb{E}}_{p}[L(f(\boldsymbol{x}), \boldsymbol{y})] = \iint p(\boldsymbol{x}, \boldsymbol{y}) L(f(\boldsymbol{x}), \boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} \tag{2}$$

However, the true distribution $p(\boldsymbol{x}, \boldsymbol{y})$ is unknown, and therefore needs to be approximated from the finite samples in the data set $\mathcal{D}$

$$\mathop{\mathbb{E}}_{p}[L(f(\boldsymbol{x}), \boldsymbol{y})] \approx \hat{\mathbb{E}}[L(f(\boldsymbol{x}), \boldsymbol{y})] \tag{3}$$

The performance of a machine learning model depends on how well it can generalize the trend seen from the limited samples in the data set to the entire input domain $\mathcal{X}$. This notion of generalization also holds true for unsupervised learning approaches, where the optimal model distribution can sample the true underlying distribution between the inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$ and generalize to new unseen data pairs.

The family of models is usually parametrized, and choosing a specific model from the model family usually means finding a configuration of the parameters which minimize the loss function. This minimization is usually performed using an optimization routine. Different training procedures are applied for unsupervised learning tasks. It should also be noted that other non-parametric approaches of machine learning do exist such as Gaussian processes [24] and Suppport Vector Machines (SVM) [22]. Although these methods are valuable in their own right, they will not be discussed further.

The next sections detail each of these critical aspects of machine learning, where model, loss, and training are elaborated upon. These aspects will be put together at the end where artificial neural networks are presented.

2.3.1 Model

A machine learning approach to problem-solving starts with a general mathematical model and uses data to adapt the model parameters to the problem being solved. A model specifies the rule or hypothesis that leads from input to output that reproduces the properties of a set of data samples. Examples of models are functions that map certain inputs to outputs, or probability distributions that sample the datapoints with high probability. It is common to separate between two types of models: *deterministic* and *probabilistic* models.

Definition (*Deterministic model*): Let $\mathcal{X}$ be an input domain and $\mathcal{Y}$ be an output domain for a supervised learning problem. A deterministic model is a function

$$y = f(\boldsymbol{x}; \boldsymbol{\theta}) \tag{4}$$

with $\boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}$, and a set of real parameters $\theta = \{\theta_1, \ldots, \theta_D\}$ (Schuld and Petricione, p.28 [16]).

For general parameters, f defines a model family. Additionally, some hyperparameters are not included in the above formalism but are often chosen a priori. An example is the way distance is measured in nearest neighbor classification tasks. In these types of tasks, a new point is assigned the label of the closest point with respect to some distance measure. Examples of such measures are absolute difference between the two points (Manhattan distance) or the absolute square difference between the points (euclidean distance). In a deterministic model, the function f acts like a function that maps inputs to outputs.

An example of such a deterministic model could be a simple linear regression in which the output of the model is given by $y = f(x; \beta_0, \beta_1) = \tilde{\beta}_1 x + \tilde{\beta}_0$ where the values of $\tilde{\beta}$ are found by training the model to fit training data.

Conversely, a probabilistic model understands the data inputs and outputs as random variables drawn from an underlying probability distribution $p(\boldsymbol{x}, y)$ where data aids the model in approximating this distribution.

Definition (*Probabilistic model*): Let $\mathcal{X}$ be an input domain and $\mathcal{Y}$ be an output domain for a supervised learning problem. Let X, Y be random variables which from which samples $\boldsymbol{x} \in \mathcal{X}, y \in \mathcal{Y}$ are drawn, and let θ be a set of real parameters. A probabilistic model refers to either the generative model distribution

$$p(\boldsymbol{x}, y; \theta), \tag{5}$$

or the discriminative model distribution

$$p(y|\boldsymbol{x}; \theta) \tag{6}$$

over the data (Schuld and Petricione, p.29 [16]).

The marginal probability $p(y|\boldsymbol{x})$ computes the probability of all possible outputs y given an input $\boldsymbol{x}$. A supervised probabilistic model can also be used to form

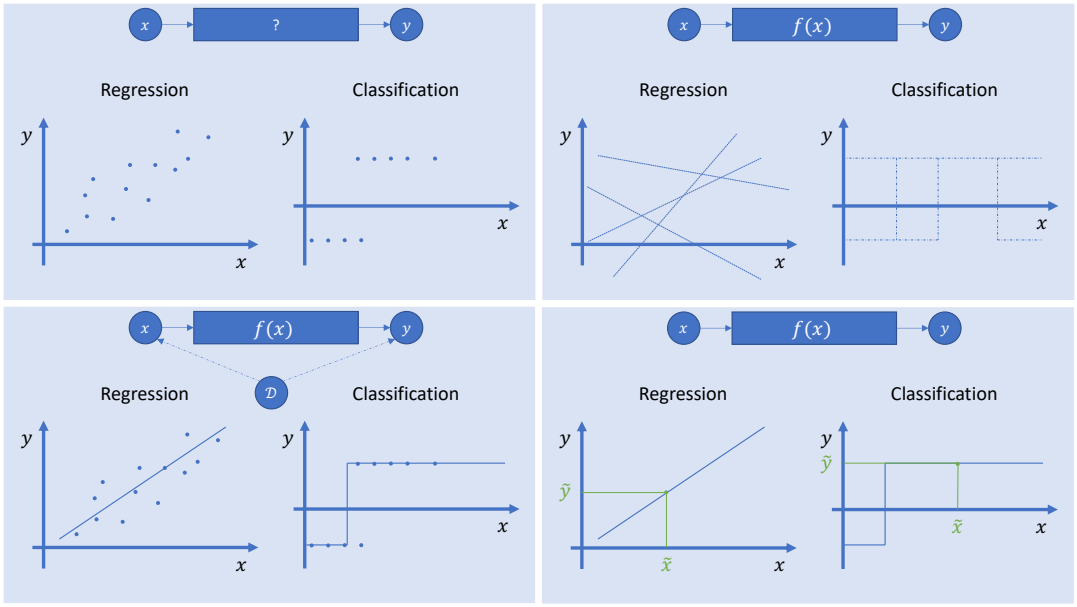


Figure 2: The four steps of a supervised machine learning approach (Adapted from Schuld and Petricione, p.31 [16]).

a deterministic model by interpreting the output of the model as deterministic. The most common practices to form a prediction from probabilistic models given a specific input $\tilde{\boldsymbol{x}}$ are to either take the maximum of the distribution or the mean of the distribution.

$$\text{Mean of the distribution: } \tilde{y} = \int dy\ p(y, \tilde{\boldsymbol{x}}) y \tag{7}$$

$$\text{Maximum a posteriori estimate: } \tilde{y} = \max_{y} p(y|\tilde{\boldsymbol{x}}) \tag{8}$$

Figure 2 illustrates how the data can turn the general model family into a model that can perform predictions. This figure assumes that the model $f(\boldsymbol{x})$ is deterministic. As the upper left figure indicates, the dataset has an unknown relationship between the datapoints (x_i, y_i). The first step is to choose which generic model family to fit the data (upper right). This choice depends on the problem at hand (linear model for the regression problem and a step-function for the classification). Training the model means choosing a specific model (or distribution) from the model family by tuning the parameters and hyperparameters of the model to estimate the data points in the training-set $\mathcal{D}$. Training a classification model means separating the input space into segments of different classes. When the model is trained to fit the data, it can be used for predicting new input variables $\tilde{\boldsymbol{x}}$.

2.3.2 Cost-function

As mentioned earlier, the goal of a supervised learning task is to minimize the risk (generalization error)

$$\mathcal{R}_{f_\theta} = \mathbb{E}\left[L_{f_\theta}\right] = \int_{\mathcal{X}\times\mathcal{Y}} L\left(f_\theta(\boldsymbol{x}), y\right) p(\boldsymbol{x}, y) d\boldsymbol{x} dy \tag{9}$$

where the integral is over all possible data pairs with $p(\boldsymbol{x}, y)$ being the true underlying distribution between the inputs and outputs, and the loss-function L being a measure of how close the predictions of the models are to target labels. However, since the true distribution is unknown, one has to minimize the empirical risk over the M samples in the data set $\mathcal{D}$ as a proxy for the real risk,

$$\hat{\mathcal{R}}_{f_\theta} = \hat{\mathbb{E}}\left[L_{f_\theta}\right] = \frac{1}{M}\sum_{m=1}^{M} L\left(f\left(\boldsymbol{x}^m\right), y^m\right) \tag{10}$$

to choose the model family with parameters that minimize this risk

$$\boldsymbol{\theta}^* = \min_{\boldsymbol{\theta}} \hat{\mathcal{R}}_{f_\theta}. \tag{11}$$

Optimal parameters in the empirical risk do not necessarily correspond to optimal parameters of the true risk. This difference means that the model performs well on data it is trained on, however for unseen instances the prediction power of the model remains low. Merely minimizing the loss function by exactly fitting a function to the training data leads to a phenomenon called *overfitting*. An overfit model suffers from learning the residual variations (e.g. noise) in the data as if they were the underlying structure. As a result, the model cannot create generalized predictions to new, unseen data. To combat overfitting, a regularizer is added to the loss function. This combination of both loss and regularizer is often referred to as the cost function

$$C(\boldsymbol{\theta}) = \text{loss} + \text{regularizer}. \tag{12}$$

As mentioned, the *loss* term measures how close the model predictions are to target variables. The loss term can take different forms depending on the type of problem that is solved. In a regression task, a typical loss function is the sum of squared error,

$$\text{Squared loss: } \frac{1}{2}\sum_{i=0}^{n}(f(\boldsymbol{x}^i) - y^i)^2 \tag{13}$$

where $(\boldsymbol{x}^i, y^i)$ are data points from the training set used to fit the model and $f(\boldsymbol{x}^i)$ is the prediction that the model makes. This specific loss function is the sum of the euclidean distances between predictions and labels in the training set.

Loss-functions for classifications take different forms since the outputs are discrete. Two examples of loss-functions used in classification tasks are the hinge loss and logistic loss functions,

$$\text{Hinge loss: } \sum_{i=0}^{n} \max(0, 1 - y^i f(\boldsymbol{x}^i)) \tag{14}$$

$$\text{Logistic loss: } \sum_{i=0}^{n} \log(1 + e^{-y^i f(\boldsymbol{x}^i)}) \tag{15}$$

These two loss-functions use the term $y^i f(x^i)$ as a distance measure since it is positive when the two numbers have the same sign and negative when they are different. These functions are useful in binary classification, where the output is $y \in \{-1, 1\}$.

There is a delicate balance between a flexible model that can predict the training data well and an overfitted model that cannot generalize from the data. The general term for preventing overfitted models is called *regularization*. There are several ways to prevent the model from overfitting with the most common being an addition of a *regularizer*-term to the cost function as done in equation 12. The regularizer is a penalty term that constraints the choice of parameters to avoid overfitting. Examples of regularizers are

$$R_{L_1}(\boldsymbol{\theta}) = \lambda ||\vec{\theta}||_1 = \lambda \sum_i |\theta_i| \tag{16}$$

$$R_{L_2}(\boldsymbol{\theta}) = \lambda ||\vec{\theta}||_2^2 = \lambda \sum_i \theta_i^2 \tag{17}$$

The L_2 regularizer adds a penalty for the length of the parameter vector, favoring parameters with a small absolute value. The L_1 regularizer favors sparse parameter vectors. A geometric way of interpreting this behavior is to consider a model that depends on two parameters (β_1, β_2). Figure 3 shows a geometric interpretation of how these two regularizers affect the optimal solution. The L_1 regularizer gives rise to a diamond-shaped constraint while L_2 norm gives rise to a circular constraint. Since the optimal parameters of the loss lie outside the constraints, the optimal parameters of the

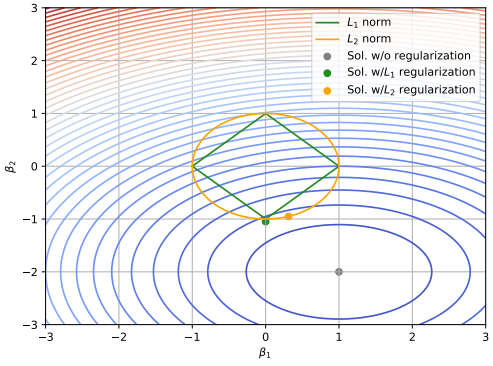


Figure 3: Plot of a loss function and the two regularizers, L_1 (green) and L_2 (orange).

cost are the intersection between the contour of the lowest loss still within the boundary of the regularizer. For the L_1 regularizer, this condition is often met at the corners of the diamond, which gives sparse solutions as shown. The hyperparameter λ regulates the regularizer's contribution to the cost. A low λ tends to give overfitted models.

In order to estimate the generalization performance of the model, one divides the available dataset into three parts: training-, validation-, and test-set. The training set is used to optimize the parameters to minimize the cost-function $C(\boldsymbol{\theta})$. The validation set is then used to estimate the performance after training to adapt hyperparameters of the model (this can be the value of λ in the regularization-term in the cost function). The test set is finally used to evaluate the model's performance on unseen data. The model's performance can be estimated by calculating the cost function on the test-set, or by using some other metric. In classification tasks, it is common to use other metrics to estimate the performance of the model, such as

$$\text{accuracy} = \frac{\text{number of correctly classified examples}}{\text{total number of examples}} \tag{18}$$

$$\text{error} = 1 - \text{accuracy} = \frac{\text{number of incorrectly classified examples}}{\text{total number of examples}} \tag{19}$$

Other measures such as false negatives and false positives might be more important to minimize in certain applications. Dividing the available data into these sets gives a way to estimate how well the model can generalize, the importance of which is highlighted in the following subsection.

2.3.3 The Bias-variance trade-off, Double Descent, and Generalization

As mentioned earlier, training only on the particulars of the training data creates the problem of overfitting, and hence regularization was introduced as a means to combat this. It is possible to show that the generalization error decomposes into three interpretable terms which shed light on how the model error typically scales with increased model complexity. For the interested reader, a derivation of the generalization error is presented in Appendix A. The resulting expression consists of three terms:

$$\text{Generalization error} = \underbrace{E_{\boldsymbol{x},D}\left[\left(f_D(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\boldsymbol{x},y}\left[(\bar{y}(\boldsymbol{x}) - y)^2\right]}_{\text{Noise}} \tag{20}$$

$$+ \underbrace{E_{\boldsymbol{x}}\left[(\bar{f}(\boldsymbol{x}) - \bar{y}(\boldsymbol{x}))^2\right]}_{\text{Bias}^2} \tag{21}$$

This expression expresses that to minimize the error, two terms must be minimized simultaneously, namely the bias and the variance of the model.

The first term expresses the variance, which measures the amount the model f_D would change if it was fit using different training data. Ideally, each model should not differ significantly between training sets, hence deviations from the average model $\bar{f}$ using different datasets should be relatively low. Flexible models that fit the data well usually give rise to high variance and are prone to overfitting since the same model family applied on two training sets gives vastly different models.

The last term represents the expected deviation of the expected model $\bar{f}$ from the expected label $\bar{y}$. Hence this term represents the inherent error associated with the model. Even with an infinitely sized data set, the model would still produce errors since the model is biased towards a certain kind of solution. For instance, applying simple linear regression to a problem where the data itself is non-linear would result in a large error since the model is too simple to capture the underlying patterns of the data set. Regardless of how many data points the model is given, the model's bias towards linear solutions would persist.

Lastly, the noise term captures the fluctuations of the labels of a newly drawn data point to the expected value of the label given the same feature-vector $\boldsymbol{x}$. Certain measures can be taken to reduce the noise by for instance increasing the number of features of the model to capture more dimensions of the underlying distribution, hence reducing the uncertainty of a given data point. However, it is generally assumed that errors are irreducible after a certain point.

Although simplifying assumptions like the squared loss and regression were made in the above derivation, it has been widely understood in statistical learning that this trend generally holds for most models. As a rule of thumb, more flexible methods generally give rise to high variance as they fit the training data well and these models have a low bias as they can capture complex trends in the data. Herein lies the trade-off. If the model is too simplistic/rigid, it cannot capture the trend of the underlying data distribution; the model is underfitting. On the other hand, if the model is too flexible, it will fit the data points of the training set very well, essentially fitting noise as part of the model; the model is overfitting. This trade-off generally gives rise to a U-shaped curve as shown in the left part of figure 4. As shown, the training error monotonically decreases with increasing model complexity since complex models can interpolate all the points in the dataset. However, the test error (generalization error) increases after a certain point as the model starts to overfit.

However, recent trends in deep learning seem to challenge this well-established trend in machine learning. Particularly, trends in deep learning suggest that once the complexity of the method reaches an interpolation threshold, the test error starts to decrease, often going beyond the lowest test error achievable from the bias-variance trade-off as illustrated in figure 4. The name interpolation threshold comes from the fact that the model is essentially interpolating all training data, hence the training error is zero. This observed phenomenon has been referred to as double descent, and little is known about its mechanisms. Belkin et al. [25] note that the traditional way of viewing a model family is flawed and that increasing the model complexity does not necessarily translate into increased variance with respect to fitting the underlying bias of the data. Instead, a more appropriate bias for the model is to

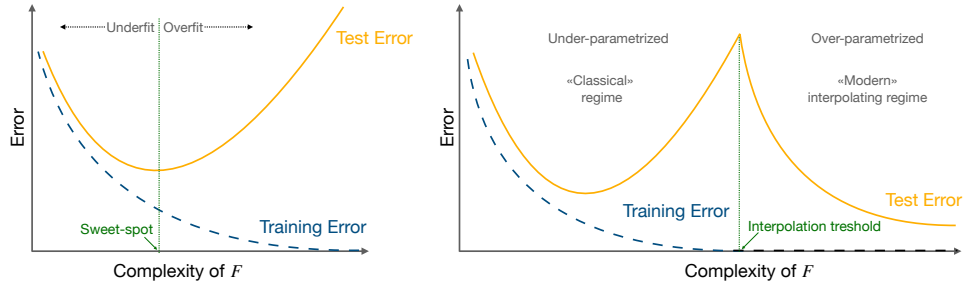


Figure 4: An illustration from the findings by Belkin et al. [25]. The left image represents the regime in which the U shape of bias-vairance trade-off operates. The right graph shows the double desecent phenomena wherein after an interpolation threshold is reached, the model generalizes increasingly well (Adapted from Belkin et al. [25]).

consider the smoothness of the function made from the model.

To illustrate the concept, consider a simple spline regression problem. At the interpolation threshold there is only one specific interpolating polynomial (of degree equal $N - 1$ where N is the number of points in the training data set) capable of interpolating all points. Therefore, the model cannot generalize well, resulting in the spike seen at the threshold. As the polynomial order increases, there are infinitely many solutions at each order that still interpolate all the points. Choosing the minimum norm solution yields a smooth behaving function that does not wildly oscillate, a form of Occam's razor since the simplest solution (least norm) is used to explain the trend in all the data points. Using this as a measure of model complexity, it is evident that the variance of a minimum norm spline regression decreases with an increase in degree.

Specifically, in deep learning, it is found that when training deep, overparametrized neural networks with stochastic gradient descent, the optimized parameters are very close to the random initial parameters. This indicates that the training routine chooses within a very limited subset of all possible models within a model family. Neglecting training and data as part of the model complexity, which effectively acts like a regularizer term, gives a misguided estimate of model complexity. It should also be noted that not all machine learning algorithms show this double descent trend, especially if a regularizer term is already built into the cost function.

2.4 Optimization

Most cost functions are non-convex and are therefore hard to optimize. As a result, most optimization algorithms rely on some form of an iterative search such as stochastic gradient descent. This method performs a stepwise search for the minimum cost on batches of data. Gradient descent updates the parameters $\boldsymbol{\theta}$ of a cost function $C(\boldsymbol{\theta})$ iteratively towards the direction of steepest descent,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla C(\boldsymbol{\theta}^{(t)}) \tag{22}$$

Each step is taken in the decreasing-cost direction. η is the learning rate that quantifies how large each step is. The problem with gradient descent is that it gets stuck in local minima (suboptimal solutions to the cost function) and becomes very slow at saddlepoints since the gradient becomes vanishingly small.

Stochastic gradient descent (SGD) uses the same procedure as regular gradient descent. However, rather than calculating the gradient using the entire dataset at each iteration, SGD approximates the gradient using a randomly selected batch of the dataset. The stochastic version has some favorable properties. Firstly, the computation needed to perform one iteration of the algorithm is smaller (which becomes significant with increasing datasets). Second, the gradient direction's stochastic nature helps escape local minima since the cost landscape is altered when using different points since the cost function itself is data-dependent.

Additional complexities can be added to the SGD-based optimizers to increase performance. For instance, the Adam (Adaptive Moment Estimation) [26] optimizer is a famous optimizer widely used in deep learning that incorporates dynamically changing learning rates and incorporates momentum. Momentum is a feature for optimizers where the parameters' change depends on the previous steps' changes.

In practice, one usually has iterative runs in which one trains the algorithms on the training data and tests the model on the test set. This iterative approach allows for the continuous monitoring of the generalization error during optimization, which means that one can truncate the training procedure once the model starts to fit the particulars of the training set. The abovementioned methods rely primarily on the gradient when navigating the cost landscape. However, when the cost function is difficult to evaluate or no gradient-functions are available, one must rely on gradient-free optimization procedures. In the following two subsections, the gradient-free optimizers Nelder-Mead and SPSA are presented in more detail as they have been widely used in the thesis.

2.4.1 Nelder-Mead

Nelder-mead is a direct search method for finding a minimum or maximum of a multi-dimensional function. Rather than evaluating the gradient to navigate the landscape, direct search methods use function evaluations in a point's vicinity to determine the direction of increasing/decreasing cost. Search methods work well, but are not guaranteed to converge.

The Nelder-Mead method method uses simplexes to navigate the cost landscape. A simplex is a shape that consist of $n + 1$ vertices in n dimensions. For instance, in two dimensions the simplex is a triangle while in three dimension it is a tetrahedron. The method begins with selecting three arbitrary points (in 2D) to form a simplex. From there the method performs a set of steps until a convergence criteria is met.

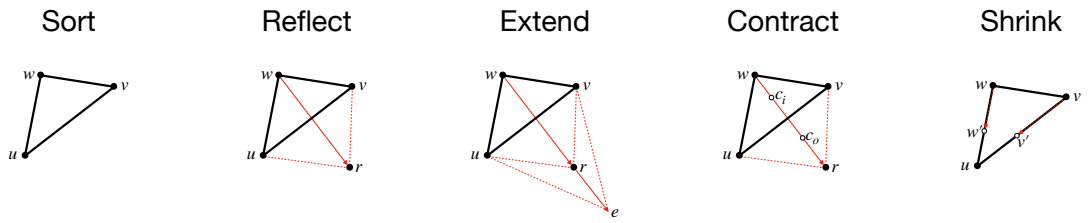


Figure 5: A schematic overview of the different steps in the Nelder-Mead procedure

These steps are sort, reflect, extend, contract and shrink. In order to demonstrate the method, these steps will be elaborated on in a two-dimensional function $f(x, y)$ where one wishes to find a local minima of this function.

The method starts with picking three points, u, v, w according to their function value where u is the best point while w is the worst point. In other words,

$$f(u) < f(v) < f(w) \tag{23}$$

Once this sorting has been established, one attempts to replace the worst point w with a better one. This is done by reflecting the point w through the line that connects the other two points u and v as shown in figure 5. In general, the reflection is performed through the centroid of the remaining points. Now, if $f(u) < f(r) < f(v)$, then w is replaced by r and the next iteration uses the points u, v, r to form the new simplex. If the reflected point r is also better than u one anticipates that this direction is one of decreasing cost. Hence a greedy extension in the direction through r is performed to the point e as shown in figure 5. If $f(e) < f(r)$, one replaces w with e, while if $f(r) < f(e)$, the point r is chosen instead.

Now, if the reflected point r is worse than both u and v and one were to accept the point, the procedure would end up in a loop since the worst point of the next iteration would again be w. Define instead points c_i and c_o as the halfway points along the reflected lines where c_i is $1/4$ between w and r while c_o is $3/4$. If either of these points perform better than the point v, w is replaced by the better point. Lastly, if neither of the contracted points outperforms v, the new simplex is shrunk towards the best performing point u, where the points v, w are moved halfway along the lines that link them to the point u. At all these steps where a different point could be chosen, the convergence is checked along some predefined convergence criteria.

2.4.2 Simultaneous Perturbation Stochastic Approximation (SPSA)

SPSA [27] is a gradient-free black box optimization procedure for multivariate functions. As with general non-gradient based methods, this optimization procedure is utilized when calculating the gradient is either not possible or costly. The SPSA optimization routine stochastically approximates the gradient using a perturbation vector $\Delta = [\Delta_1, \Delta_2, \ldots, \Delta_p]^T$ and calculates the gradient, $\boldsymbol{g}_k(\boldsymbol{\theta}_k)$, using only two function evaluations

$$\boldsymbol{g}_k\left(\boldsymbol{\theta}_k\right) = \frac{y\left(\boldsymbol{\theta}_k + c_k\Delta_k\right) - y\left(\boldsymbol{\theta}_k - c_k\Delta_k\right)}{2c_k} \begin{bmatrix} \Delta_{k1}^{-1} \\ \Delta_{k2}^{-1} \\ \vdots \\ \Delta_{kp}^{-1} \end{bmatrix}. \tag{24}$$

This approximated gradient contrasts the $2p$ shifts used for a finite difference approximation for a parameter-vector of p parameters. The perturbation vector is randomly generated at each step, with the components generated from a zero-mean distribution. Very commonly the components are sampled from the Bernoulli distribution and hence each parameter is simultaneously perturbed by either $\pm c_k$. The update rule is similar to that of SGD, namely

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - a_k \boldsymbol{g}_k(\boldsymbol{\theta}_k) \tag{25}$$

$$a_k = \frac{a}{(A+k+1)^\alpha}, \quad c_k = \frac{c}{(k+1)^\gamma} \tag{26}$$

however in contrast to SGD, the step size continually decreases with each iteration k. The performance of this optimization procedure heavily depends on five free hyperparameters a, c, α, γ, A. Spall [27] provides some guidelines to choosing these hyperparameters, namely that $\alpha = 0.101, \gamma = 0.602$ and that A should be roughly 10% of the total number of iterations used in the procedure. The remaining two parameters must be chosen based on the problem the procedure is being applied to. a essentially acts like the learning rate while c is the scaling of the random shift.

SPSA is often applied to noisy optimization problems because of the stochastic perturbation vector. Since each parameter is already being perturbed, additional shifts from noise has less of an effect.

To summarize, a general method in machine learning specifies a general model family of *functions* (deterministic) or *distributions* (probabilistic) which is useful for the prediction of new inputs. Additionally, a training strategy is needed to use the data to construct a specific model which can generalize from the given training data. The overarching goal of the learning algorithm is to minimize the generalization error, which is the error made on new data instances. All machine learning algorithms follow these general underlying steps, however, some nuances differentiate the machine learning methods. The methods most relevant for quantum machine learning can be separated into four categories: data fitting, artificial neural networks, graphical models, and kernel methods. Rather than introducing all these methods, this thesis will reduce the scope to neural networks.

2.5 Artificial Neural Networks

Artificial neural networks can be seen as a non-linear model in which an input vector is processed through layers of different non-linear transformations. This model has its inspiration from biology in which information is fed to the brain through layers of firing neurons. The basic building-block of neural networks are often referred to as perceptrons and are given by

$$\varphi(x) = \begin{cases} 1, & \text{if } \boldsymbol{w}\boldsymbol{x} \geq b \\ -1, & \text{else} \end{cases} \tag{27}$$

where $\boldsymbol{w}$ is a vector of trainable weights and b is the bias. The bias term shifts the decision boundary from the origin. Once this threshold is reached, the perceptron "fires". Mathematically, the perceptron is a linear classifier that classifies inputs into a class of outputs based on an activation function.

Neural networks are often referred to as multi-layer perceptrons. In these models, the output from one perceptron is the input to another. Individual neurons may have different non-linear activation functions that cause them to fire differently from the non-linear activation function of the original perceptron. There are multiple types of neural networks, some common ones being feedforward neural networks, recurrent neural networks, and Boltzmann machines.

2.5.1 Feedforward Neural Networks

Feedforward neural networks are deterministic models, and their general form is given by (with bias term included in W_i)

$$f(x, W_1, W_2, \dots) = \phi_N(W_N \phi_{N-1}(\dots \phi_2(W_2 \phi_1(W_1 \boldsymbol{x}))). \tag{28}$$

Each layer in a feedforward neural network is associated with a vector wherein each vector element represents a node in that layer. There are three different types of layers: an input layer, an output layer, and hidden layers. The input layer $\boldsymbol{x}$ consists of decoding data into the features that are used in the model. This layer is connected to the first of L hidden layers, $\boldsymbol{h^l}$ where $l \in \{1, \dots, L\}$. These hidden layers are used for finding patterns in the training data, and each hidden layer is associated with different activation functions $\phi_i(x)$. The activation function is applied element-wise. Examples of widely used activation functions are $\tanh(x)$, ReLU and the sigmoid function,

$$\text{ReLU} : \varphi(x) = \begin{cases} a \text{ if } x > 0 \\ 0 \text{ else.} \end{cases}, \quad \text{Sigmoid} : \varphi(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{29}$$

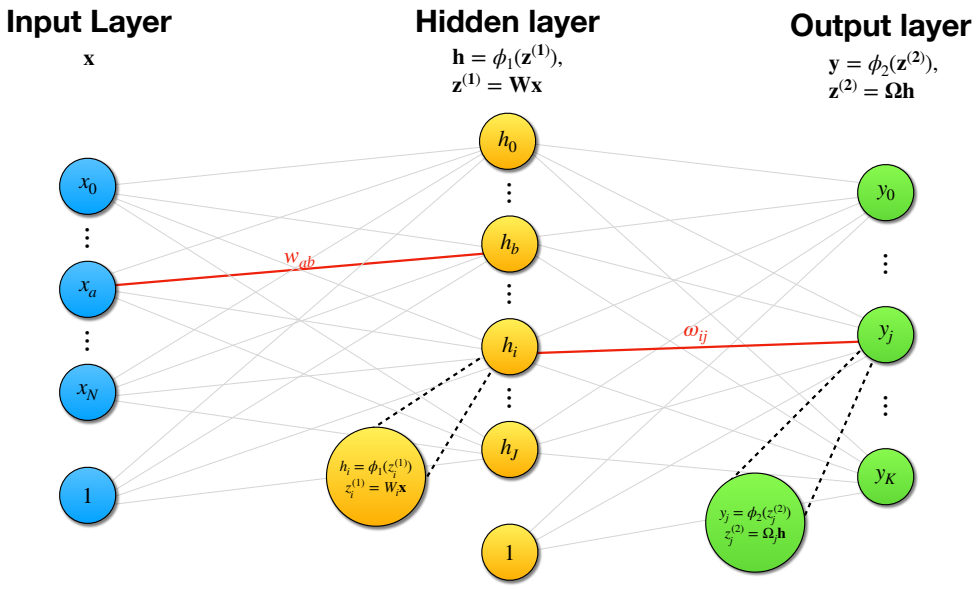


Figure 6: A neural network with one hidden layer. The illustration also includes how a unit processes information from the previous layer, and is highlighted in nodes h_i and y_j. The notation for the variables is described in the backpropagation section (Adapted from Schuld and Petricione, p.53 [16]).

The last hidden layer $\boldsymbol{h^l}$ is connected to the output layer which returns a vector $\boldsymbol{y}$. This vector is typically associated with a classification in which each element is the probability of a certain classification. The largest element is commonly assumed to be the prediction of the model.

A node in one layer is linearly dependent on all the nodes from the previous layer, but the contribution from each of the previous layers' nodes is weighted differently. This can be represented by an edge in the neural network, as shown in figure 6 with varying weights. This linear dependence between one node and the previous layer can be represented by a matrix W_i, where each element w_{ab} signifies the weight of node b in the previous layer's contribution to node a in the current layer. Additionally, by attaching a column vector at the end of the weight matrices and including a node at each layer set to the value 1, one may include terms often referred to as biases. These trainable constants do not couple to the nodes in the previous layer. The non-linearity of these architectures comes from the fact that the output from the node undergoes the non-linear activation function. Therefore, the output from one layer is $\boldsymbol{a^i} = \phi(W\boldsymbol{a^{i-1}})$ where $\boldsymbol{a^i}$ denotes a layer. Recursively performing this procedure from the output layer until the input-layer results in equation 28.

2.5.2 Backpropagation

The trainable parameters of the models are the weights of the matrices that connect two layers. Training these parameters are done using gradient descent methods; therefore the gradient of the loss function with respect to each weight has to be calculated. These weights are computed using the chain rule. Backpropagation

is efficient because one avoids redundant calculations of intermediate terms in the chain rule. This redundancy is avoided by starting from the outer-most layer and computing the gradient iteratively backward. To see how calculating the gradient backward layer by layer reduces the needed calculations, consider the following.

Consider a neural network with a N-dimensional input-layer, one J-dimensional hidden layer, and a K-dimensional output layer. Connecting the input and hidden layers is the weight matrix $W_1 = W = \mathbb{R}^{J\times N}$ while the hidden layer and output layer is connected by $W_2 = \Omega = \mathbb{R}^{K\times J}$. Each element of these matrices is given by w_{ij} and ω_{ij}. These matrix elements connect node i in the current layer to node j in the right neighboring layer. Additionally, W_k and Ω_k describe the k-th row of these matrices. $z_k^{(1)} = W_k\boldsymbol{x}$ and $z_j^{(2)} = \Omega_j\boldsymbol{h}$ describe the linear dependence between the k-th node in a layer and all the nodes in the previous layer. Figure 6 shows a schematic view of this neural network with the relevant variables. In order to perform gradient descent to update each of the weights, two derivatives are needed, namely $\frac{\partial C}{\partial \omega_{ij}}$ and $\frac{\partial C}{\partial w_{ij}}$ for each of the matrices Ω and W respectively. In figure 6 the indices are separated between ab and ij to show that these sets of indices are independent. The derivatives can be calculated using the chain rule as follows:

$$\frac{\partial C}{\partial \omega_{ij}} = \frac{\partial C}{\partial y_j}\underbrace{\frac{\partial y_j}{\partial z_j^{(2)}}}_{\phi_2'(z_j^{(2)})}\underbrace{\frac{\partial z_j^{(2)}}{\partial \omega_{ij}}}_{h_i} = \delta_{y_b} h_i \tag{30}$$

As evident by the expression, updating a single hidden-to-output layer weight ω_{ij} requires the gradients of the output unit y_j that it leads to. Similarly, the derivative of the cost with respect to the weights between the input and hidden layer is given by

$$\frac{\partial C}{\partial w_{ab}} = \sum_{k=1}^{K}\underbrace{\frac{\partial C}{\partial y_k}\frac{\partial y_k}{\partial z_k^{(2)}}}_{\delta_{y_k}}\underbrace{\frac{\partial z_k^{(2)}}{\partial h_b}}_{\omega_{bk}}\underbrace{\frac{\partial h_b}{\partial z_b^{(1)}}}_{\phi_1'(z_b^{(1)})}\underbrace{\frac{\partial z_b^{(1)}}{\partial w_{ab}}}_{x_a} \tag{31}$$

$$= \underbrace{\left(\sum_{k=1}^{K}\delta_{y_k}\omega_{bk}\right)\phi_1'(z_b^{(1)})}_{\delta_{h_b}}\, x_a \tag{32}$$

$$= \delta_{h_b} x_a \tag{33}$$

Updating the input-to-hidden layer weight requires the gradient of all output units ($\sum_{k=1}^{K}\delta_{y_k}\omega_{bk}$) and the hidden unit it leads to ($\phi_1'(z_b^{(1)})$). An important notion in this expression is the sum over the K nodes in the output layer. This is because all the nodes in the output layer depend on h_b, so a small change in this specific node would generally propagate to all the nodes connected to it. Including more hidden layers is simple; let the sum run over all nodes in all layers in front of the

layer the weight leads to. The method is called backpropagation since the error terms, δ, are propagated backward in the neural network to calculate the derivatives. Backpropagation employs dynamical programming since the nodes' derivative within a layer depends on the errors of the layers in front of it. Note also that the weight derivatives have the same dimensions as the weight matrix. Therefore, the computations in backpropagation are highly parallelizable and widely used in deep neural networks.

Classical post-processing of measurements from quantum hardware using Neural Networks has shown great promise, primarily because of the efficiency of backpropagation. Particularly, it is found that training these classical models is significantly easier than training quantum models; the reason why will be elaborated on in section 4. This interplay between Neural Networks and VQAs will be explored in section 7. However, an introduction to quantum machine learning is needed to understand VQAs and their parallels to machine learning.

3 Quantum Machine Learning

The term Quantum Machine Learning has no definite definition. It is often widely used as an umbrella term for computing methods that utilize some form of both quantum resources and traditional machine learning approaches. Schuld and Petruccione [16] divides the field into four categories based on how the data is generated (classical or quantum) and how the information is processed (classical or quantum). This separation is presented in figure 7. Since both the field of quantum computing and machine learning are in development, this framework is not rigorous and labeling all aspects of the interplay between the disciplines is difficult. Therefore, this framework is primarily a guide that indicates aspects of the interplay rather than a rigorous definition that consistently applies to all quantum machine learning models. Confusion arises primarily when the differentiation between quantum and classical becomes blurry.

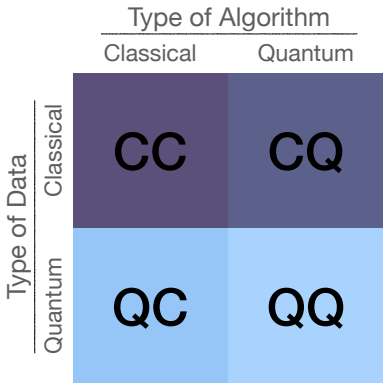


Figure 7: The four combinations of quantum computing and machine learning. The first letter stands for the type of system while the second letter represents the information processing device (Adapted from Schuld and Petruccione, p.6 [16]).

Classical data refers to data that can be represented on a classical computer while quantum data traditionally refers to data that is inherently quantum, like a quantum state. Confusion in the labeling of data as quantum or classical arises when considering a data-generating device that is quantum, like a molecule. If the exact quantum state of the molecule is the input to some other processing device, the input data would be labeled quantum. However, if repeated measurements are performed on the molecule which was then processed, one may argue that this is classical data since measurement outputs from a quantum computer is a bitstring. At that point, the quantum state is essentially viewed as an underlying probability distribution from which classical sampling procedures are performed.

Classical algorithms refer to machine learning approaches computed on classical hardware while quantum algorithms are those performed on quantum hardware. Processing is inherently different on quantum hardware as the properties of entanglement and superposition from quantum mechanics introduce novel methods unavailable through classical machine learning methods. Although the differentiation between quantum and classical algorithms is fairly straightforward, confusion arises in hybrid computational models. In these models, the outputs of a quantum procedure are the input to a classical machine learning routine or vice versa. As a result, it is difficult to label the entire processing device as one or the other.

This master thesis explores the QC interplay further. In particular, optimization routines applied in classical machine learning will aid in training a type of quantum

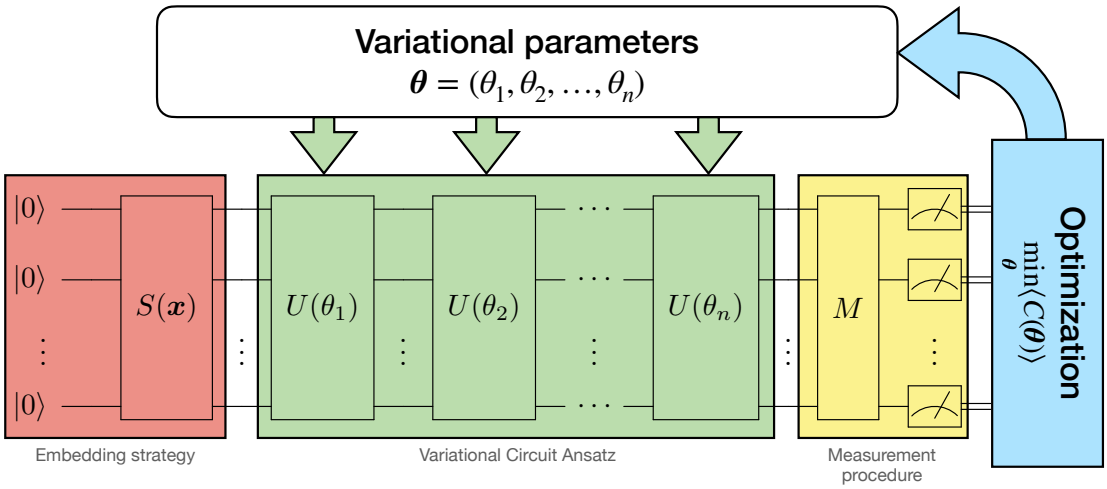


Figure 8: The three parts of a general variational circuit. The structure starts with embedding classical information into a quantum state performed by a unitary $S(\boldsymbol{x})$. The state evolution is conducted using a series of unitaries, which are variational for VQAs. Lastly, measurements are conducted, which may involve the transformation of eigenbasis. In the case of a variational circuit, a classical optimization procedure is used to find the optimal parameter that minimizes some cost function.

algorithm called Variational Quantum Algorithms (VQA). This paper labels this interplay QC because a variational circuit is used to evolve a quantum state, from which computational basis measurements are sampled. These measurements are the inputs to a classical optimizer that optimizes the variational parameters of the quantum circuit, which creates a feedback loop as shown in figure 8. In order to incorporate the notion that VQAs use quantum processing of information, the paper labels this interplay as QC and not CC, even though the information into the classical processing device is inherently classical as alluded to earlier. Additionally, Schuld (2021, p.8) [16] refers to the QC intersection as "how machine learning can help quantum computing" which reflects this work's use of machine learning best. This interplay will be explored further by post-processing the measurement data using a classical neural network, elaborated on in section 7.

A general quantum routine consists of three parts: state initialization, state preparation, and measurements as shown in figure 8. These three processes are different depending on the problem definition and how the algorithm is structured. For instance, state initialization may involve simple routines such as initializing the $|+\rangle^{\otimes N}$ state ($|+\rangle$ is one of the eigenstates of the Pauli X matrix), as is the case in certain VQAs, or may require the embedding of classical data into quantum states. Examples of embedding strategies are presented in section 3.2. The solutions to problems are also different, primarily reflected in the measurement procedure. Consider for instance solving combinatorial optimization problems on quantum hardware. These problems require repeated measurements of the prepared quantum state in the computational basis, i.e the eigenbasis of the Pauli Z operator. On the other hand, chemistry problems may require the measurement of some arbitrary Hamiltonian. How the measurement of arbitrary Hamiltonians is performed in a mathematical sense is presented in section 3.3.

The most differentiating factor between different types of quantum routines is the preparation of the quantum state, which is a unitary evolution of the initial state through a quantum circuit. Long-term quantum algorithms such as Shor's algorithm [5], Grover search [28], and the Quantum Fourier Transform [29], employ gates that remain fixed in their structure. In contrast, the near term algorithms primarily employ parametrized circuits in which parameters of the gates are varied. This project assignment examines algorithms involving circuits of the latter type. This section starts by presenting the general structure of VQAs.

3.1 Variational Quantum Algorithms (VQA)

Variational Quantum Algorithms is a class of hybrid quantum-classical algorithms. The quantum computer prepares a quantum state with variational parameters, while the classical computer optimizes the parameters to minimize some cost function. This hybrid approach directly addresses some of the issues present in NISQ devices. Since the optimization of the parameters is offloaded to the classical computer, short variational circuits are still capable of exploring sizable parts of the Hilbert space. Instead of running long circuits, this iterative approach instead runs short circuits several times, exploiting the qubits' short decoherence times and limiting the gate-induced noise of larger circuits. Other hardware limitations, such as limited qubit connectivity, are also considered when designing the structure of the quantum circuits.

The inputs of a VQA are a training set, a cost function, and an ansatz. The cost-function, $C(\boldsymbol{\theta})$ encodes the problem's solution based on the problem description and the available training data. The ansatz is a quantum circuit that contains a set of parametrized gates. These parameters are trained to minimize the cost. The training of these parameters (using training data) is performed in a quantum-classical hybrid loop to solve the optimization task.

In essence, the goal of a VQA is similar to the machine-learning approaches mentioned earlier in this paper. Both attempt to find a set of parameters through learning-based approaches which minimize some cost,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta})) \tag{34}$$

This quantum-classical hybrid loop consists of two parts. The first part evaluates the cost function (or its gradients) on quantum hardware. This involves preparing a variational state $|\psi\rangle = U(\boldsymbol{\theta})|\psi_0\rangle$ based on a circuit ansatz and performing measurements of this prepared state. This information is used in a classical optimization routine for training to gradually approach a smaller cost corresponding to a better solution to the problem. The output of the VQA is task-dependent, which is reflected by the cost function.

3.1.1 Cost function

The cost function in VQA is similar to those introduced in the machine learning part of the thesis. Mapping the trainable parameters to real-valued costs creates a hypersurface (cost landscape) as a function of the trainable parameters $\boldsymbol{\theta}$. The classical optimizer navigates this landscape to find the global minimum.

The cost function should be *faithful, efficiently estimatable, operationally meaningful*, and *trainable*. In short, this means that the optimal solution of the cost function should reflect the desired solution to the problem and that smaller cost values reflect a better solution. Additionally one should be able to efficiently estimate the cost function on a quantum computer through measurements (preferably not efficiently computable on a classical computer) and some classical post-processing. Lastly, the cost should be trainable to optimize the circuit parameters $\boldsymbol{\theta}$.

Due to the faithful nature of the cost function, different problems have different cost functions. The standard choice of the cost function is the one used in Variational Quantum Eigensolvers (VQE). This particular VQA is widely used in chemistry and condensed matter applications to find the ground state energy and state. The cost function of these circuits is

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle, \quad |\psi(\boldsymbol{\theta})\rangle = MU(\boldsymbol{\theta})S(\boldsymbol{x})|\psi_0\rangle \tag{35}$$

where $|\psi_0\rangle$ is an easily initializable state (usually $|0\rangle^{\otimes N}$), $S(\boldsymbol{x})$ is a circuit that encodes classical information into a quantum state, $U(\boldsymbol{\theta})$ is a variational circuit which evolves this state and H is the Hamiltonian of the problem which is to be solved. Measuring a Hamiltonian on quantum hardware requires a change of basis, represented by the unitary M. A visual representation of this total unitary evolution was presented earlier in figure 8. Although the trainability of this cost function is dependent on the depth of the circuit, notice that this cost function realizes the four above criteria of a cost function. This is due to the variational method of quantum mechanics, namely $\frac{\langle\Psi|H|\Psi\rangle}{\langle\Psi|\Psi\rangle} \geq E_0$ where E_0 is the ground state energy and $|\Psi\rangle$ is an unknown wavefunction.

Alternatively, the expectation value can be post-processed as input to another function. For instance, some research has been proposed to create supervised quantum machine learning models for classification tasks [30], inspired by the supervised machine learning models mentioned earlier. A natural extension of the squared loss function to include a quantum observable would be the following cost:

$$C(\boldsymbol{\theta}) = \sum_i [y^{(i)} - \langle\psi_0|S^\dagger(\boldsymbol{x}^{(i)})U^\dagger(\boldsymbol{\theta})M^\dagger HMU(\boldsymbol{\theta})S(\boldsymbol{x}^{(i)})|\psi_0\rangle]^2. \tag{36}$$

As before, the pair $(\boldsymbol{x}^{(i)}, y^{(i)})$ represents pairs of data points from the classical training set. As seen from both these cost functions, the result of the algorithm is dependent on the embedding of classical information into quantum states through the

unitary transformation $S(\boldsymbol{x})$, the measurement procedure M to perform measurement of an observable H and the state evolution $U(\boldsymbol{\theta})$. In the remaining subsections, detailed explanations of each aspect of a general VQA will be provided, starting with encoding classical data into quantum states.

3.2 Encoding Strategies

As in machine learning algorithms, it is crucial to get a representation of the data that will be trained on. There are multiple ways of encoding classical data into a quantum state. Different strategies give different properties of the quantum state, and different algorithms use different strategies. This section will give a brief overview of basis and amplitude encoding.

3.2.1 Basis Encoding

Basis encoding is an encoding strategy where one encodes binary information into an n-qubit computational basis state. For example, encoding the number 5 into a 3-qubit quantum state would result in the state $|101\rangle$. In many ways, the amplitude of the basis encoded state carries with it information about the output of an algorithm. For instance, if the absolute square of the $|101\rangle$ amplitude, $|\alpha_{101}|^2$, is larger than 0.5, then repeated circuit measurements will sample this state with higher probability than the rest of the states. In this case, 101 would be interpreted as the algorithm's output. A critical element of basis encoded algorithms is increasing the probability of measuring the basis states corresponding to correct solutions to a problem.

The positive aspect of this encoding strategy is that all classical systems encode information in binary, hence encoding information merely initializes qubits in the needed bit configuration. However, in contrast to amplitude encoding strategies, this strategy requires the same number of qubits as the classical bits-representation of information.

3.2.2 Amplitude encoding

This strategy is about encoding the information of a classically normalized vector or matrix into the amplitudes of the quantum state vector,

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix} \leftrightarrow |\psi_x\rangle = \sum_{j=1}^{2^n} x_j |j\rangle, \quad \sum_k |x_k|^2 = 1 \tag{37}$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \leftrightarrow |\psi_A\rangle = \sum_{i=1}^{2^m} \sum_{j=1}^{2^n} a_{ij} |i\rangle|j\rangle, \quad \sum_{ij} |a_{ij}|^2 = 1 \tag{38}$$

where the index registers $|i\rangle|j\rangle$ corresponds to the ith row and jth column of the matrix A. Fixing one of these registers can address the row or column of the matrix. If A is a Hermitian positive trace-1 matrix, one can associate A's entries with the density matrix entries so that $a_{ij} = \rho_{ij}$.

This type of encoding is more qubit efficient than basis encoding. It is easier to see why by working through an example. Consider the embedding of the classical number 15 into a quantum state. Converting 15 into a 4-bit binary number gives 1111. If the number was basis encoded, the corresponding state would be the 4-qubit state $|1111\rangle$.

In amplitude encoding one operates with an amplitude vector, which for a 2-qubit state would be $\alpha = [\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}]$ which represents the state

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{39}$$

The superposition of computational basis states is also a valid state, so given that the amplitudes are normalized, this is a valid quantum state. Now, to encode the binary string 1111 into a quantum state, one first normalizes it and assigns the elements of the amplitude vector equal to the normalized string. Hence, $\alpha = \frac{1}{2}[1, 1, 1, 1]$ and gives the resulting state

$$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle. \tag{40}$$

With three qubits there are 8 elements in the amplitude vector, with four qubits there are 16 elements in the vector and with five there are 32. A superposition of computational basis state is also a valid quantum state. As a result, the qubit requirement for preparing a state scales logarithmically in the bits needed to classically represent the same state. So a classical vector with N entries can be encoded in $\log_2(N)$ number of qubits.

However, even though this encoding procedure results in an exponentially smaller representation of classical information, it is not widely used in quantum algorithms since encoding information into a probabilistic description of a quantum system poses limitations on the possible executions. In particular, it is impossible to perform a nonlinear map of the amplitudes using only unitaries since nonlinear operators would violate fundamental principles of quantum mechanics. Additionally, initializing such a state is costly; hence, this encoding strategy is not viable for near-term applications.

Conclusively, note that when employing these different encoding strategies the circuit needed to embed classical information into quantum states can become large. Therefore, the encoding procedure must be considered when QML algorithms' runtime is studied. In contrast, traditional machine learning algorithms do not consider data representation and embedding a part of the runtime. Most algorithms for near-term

applications rely on basis encoding since circuits for other embedding strategies become too deep for NISQ devices. Unless specified, the encoding strategy used in the remainder of the thesis will be basis encoding. Table 1 gives an overview of other possible encoding strategies that provide different properties which may be useful for different purposes.

3.3 Quantum Measurement

One of the postulates of quantum mechanics states that a measurement of a quantum state collapses the wave function into an eigenstate of the observable and one measures the corresponding eigenvalue of the operator. Although different methods are possible, qubits were traditionally made using electron spin, where spin up and down gives rise to a two-level state used to perform computations. The spin up and down states are the eigenbases of the Pauli Z operator,

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{41}$$

Measuring the observable Z on the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ leaves two possibilities: measure the eigenvalue 1 with probability $|\alpha|^2$ or -1 with probability $|\beta|^2$. Here $|0\rangle$ and $|1\rangle$ are the eigenvectors of the Z matrix. Practical limitations of quantum hardware only restrict measurements in the Pauli Z basis, hence a measurement outcome in the computational basis is a bit string of 0 and 1. If other observables

Classical data	**Properties**	**Quantum State**
Basis encoding		
$(b_1, \ldots, b_d), \quad b_i \in \{0, 1\}$	b encodes x $\in \mathbb{R}^N$ in binary	$\|x\rangle = \|b_1, \ldots, b_d\rangle$
Amplitude encoding		
$x \in \mathbb{R}^{2^n}$	$\sum_{i=1}^{2^n} \|x_i\|^2 = 1$	$\|\psi_x\rangle = \sum_{i=1}^{2^n} x_i \|i\rangle$
$A \in \mathbb{R}^{2^n \times 2^n}$	$\sum_{i=1}^{2^n} \sum_{j=1}^{2^n} \|a_{ij}\|^2 = 1$	$\|\psi_A\rangle = \sum_{i=1}^{2^n} \sum_{j=1}^{2^n} a_{ij} \|i\rangle\|j\rangle$
$A \in \mathbb{R}^{2^n \times 2^n}$	$\sum_{i=1}^{2^n} a_{ii} = 1, a_{ij} = a_{ji}^*$	$\rho_A = \sum_{ij} a_{ij} \|i\rangle\langle j\|$
Qsample encoding		
$p(x), x \in \{0, 1\}^{\otimes n}$	$\sum_x p(x) = 1$	$\sum_x \sqrt{p(x)} \|x\rangle$
Dynamic encoding		
$A \in \mathbb{R}^{2^n \times 2^n}$	A is unitary	U_A with $U_A = A$
$A \in \mathbb{R}^{2^n \times 2^n}$	A is hermitian	H_A with $H_A = A$
$A \in \mathbb{R}^{2^n \times 2^n}$	-	$H_{\tilde{A}}$ with $\tilde{A} = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}$

Table 1: A short summary of how different types of classical data can be encoded into a quantum state (Adapted from Schuld and Petruccione, p.109 [16]).

are to be measured, a unitary transformation must be applied to the qubits prior to measurements that map measurements in the computational basis into measurements in the basis of the observable. If the hardware supports measurements in the computational basis $\{|k\rangle\}$, but a measurement in the basis $\{|\phi_k\rangle\}$ is needed, the general unitary transformation

$$M = \sum_k |k\rangle\langle\phi_k| \tag{42}$$

can map from the computational basis into the desired basis. This means that measuring $M|\psi\rangle$ in a particular computational basis state $|j\rangle$ is equivalent to measuring $|\psi\rangle$ in the $\{|\phi_k\rangle\}$ basis:

$$\big|\langle j|M|\psi\rangle\big|^2 = \big|\sum_k \langle j|k\rangle\langle\phi_k|\psi\rangle\big|^2 = \big|\langle\phi_j|\psi\rangle\big|^2 \tag{43}$$

For a single qubit, this transformation can be interpreted as rotating the Bloch-sphere so the observable's eigenvectors align with the z-axis. For general multi-qubit systems, a measurement is performed by rotating the eigenvectors of the operator to align with the standard Z-basis vectors. Therefore, this means that measuring in the Z-basis collapses the qubit-state onto eigenvectors of the operator of interest.

Measuring the eigenvalues of any arbitrary Hamiltonian (which is often the case in quantum machine learning) by creating the above unitary using circuits is either intractable or requires circuits of high depth. Therefore, it is standard practice to decompose the Hamiltonian of interest into a series of Pauli strings,

$$H_P = \sum_i P_i, \quad P_i = \{X, Y, Z, I\}^{\otimes N} \tag{44}$$

$$\langle H_p\rangle = \sum_i \langle P_i\rangle \tag{45}$$

Pauli strings are essentially concatenated Pauli-matrices across multiple qubits (mathematically a Pauli string is a tensor product of multiple Pauli-matrices), where the remaining two Pauli matrices X, Y are given by

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{46}$$

The eigenvectors of X are usually referred to as $|+\rangle$ and $|-\rangle$. On the Bloch-sphere, these vectors lie along the X-axis. Similarly, the eigenvectors of Y lie along the Y direction on the Bloch-sphere and are notated with the vectors $|i\rangle$ and $|-i\rangle$. To

measure the X, Y bases, the Bloch sphere has to be rotated such that these axes align with the Z axis. Two gates, the Hadamard (H)- and the S-gates, are needed for these transformations which are given by

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{47}$$

Through explicit matrix multiplication it can be readily shown that

$$HZH = X, \quad SHZHS^\dagger = Y \tag{48}$$

Hence, from these expressions, it is evident that measuring in the Z basis after the unitary transformations $H, HS^\dagger$ will result in measurements of the X, Y observables respectively. Because of these simple transformations, Pauli strings are widely used in VQAs since it exchanges having too many gates with having many measurements. Although the number of measurements might pose practical limitations on the computations, Gokhale et al. [31] show that by exploiting commutative relationships between Pauli strings it is possible to perform simultaneous measurements of Pauli strings. Appendix B presents this algorithm for the interested reader. This algorithm has practical importance for VQE applications of finding the ground state energy of a problem Hamiltonian. The problems in the master thesis are of diagonal Hamiltonians; hence, this procedure is unnecessary.

With encoding strategies and measurement procedures described, the most crucial aspect of VQAs remain, namely the unitary evolution of a quantum state, $U(\boldsymbol{\theta})$. The unitary evolution follows a fixed circuit structure with variational parameters. These circuit structures are referred to as circuit ansätze and are the topic of the next section.

3.4 Circuit Ansatz

Different VQAs use different sequences of gates to evolve the initial state into a final state used for measurements. The circuit ansatz defines the base structure of the gates to be applied, wherein some of the gates are associated with variational parameters while others are not. As a result, the number of changeable parameters $\boldsymbol{\theta}$ in the variational quantum model is heavily dependent on the choice of circuit ansatz. As with Neural Networks there is "no one size fits all"-type of circuit ansatz, as all ansätze have their strengths and weaknesses. The general form of an ansatz is a sequence of L applied unitaries

$$U(\boldsymbol{\theta}) = U_L(\boldsymbol{\theta_L}) \cdots U_2(\boldsymbol{\theta_2}) U_1(\boldsymbol{\theta_1}) \tag{49}$$

$$U_l(\boldsymbol{\theta_l}) = \Pi_m e^{-i\theta_m H_m} W_m \tag{50}$$

Each U_l is a unitary layer consisting of a sub-routine of m gates that are either parametrized ($e^{-i\theta_m H_m}$) with H_m being a Hermitian operator or unparametrized (W_m). The circuit ansätze come in two overarching forms: Problem-specific and Problem-Agnostic ansätze. Problem-specific ansätze use the problem's structure to tailor the ansatz, while problem-agnostic ansätze consider no aspects of the problem in their design.

3.4.1 Problem-agnostic ansätze

Hardware-efficient ansätze: Problem-agnostic ansätze are ansätze that are constructed using no information about the problem Hamiltonian. Restrictions in quantum hardware, such as limited qubit connectivity and multi-qubit gates, form the design of these ansätze. For instance, if the quantum hardware only allows for two-qubit gates and only nearest-neighbor interactions, only certain entanglement procedures are possible. This restriction limits the applicable quantum gates to three-qubit gates between the qubits' nearest neighbors. Advantages with these types of circuits are that Hamiltonians with similar structures to the device interactions can be readily studied. Also, since these ansätze are designed with the device structure in mind, it avoids some of the overhead needed to translate an arbitrarily designed circuit to the implementable gates on hardware. Since these ansätze use no information about the Hamiltonian of the problem, these types of ansätze can be applied to various problems.

3.4.2 Problem-specific ansätze

Unitary coupled-cluster ansatz: This ansatz is used in quantum chemistry problems to obtain the ground state of a fermionic molecular Hamiltonian H. The ansatz takes the form

$$e^{T(\theta)-T(\theta)^\dagger}|\psi_0\rangle, \tag{51}$$

$$T = \sum_k T_k, \quad T_1 = \sum_{i,j} \theta_i^j a_i^\dagger a_j, \quad T_2 = \sum_{i,j,k,l} \theta_{i,j}^{k,l} a_i^\dagger a_j^\dagger a_k a_l \tag{52}$$

In these ansätze, the $|\psi_0\rangle$ is usually the Hartree-Fock state of H; an approximation of the ground-state wavefunction commonly used in computational chemistry [32]. This state is a good starting point for finding the system's ground state since it is most likely close to the Hartree-Fock state. T is the cluster-operator which is commonly truncated at single excitations T_1 and double excitations T_2 and $\{a_i, a_i^\dagger\}$ are fermionic destruction and creation operators. Qubits on the other hand operate with spin-Hamiltonians, and therefore a mapping from fermionic operators to spin-operators is needed, often using the Jordan-Wigner mapping [33].

Quantum alternating operator ansatz (QAOA): The QAOA ansatz is inspired by adiabatic quantum computing; a type of computing where the input state $|\psi_0\rangle$ is

adiabatically transformed into the ground state of a problem Hamiltonian H_p. This transformation happens by gradually turning off the initial Hamiltonian H_0 (which gave rise to $|\psi_0\rangle$) and gradually introducing the problem hamiltonian H_P. The QAOA ansatz deviates from this approach, where instead of gradually introducing the problem Hamiltonian into the system, it instead alternatively applies a problem unitary (involving the problem Hamiltonian H_P) and a mixer unitary (involving a mixer Hamiltonian H_M with the condition $[H_P, H_M] \neq 0$). The total unitary for this ansatz is given by

$$U(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \Pi_{l=1}^{p} e^{-i\beta_l H_M} e^{-i\gamma_l H_P}, \quad \boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\beta}) \tag{53}$$

The number p refers to the depth of the ansatz. It describes how many layers of this alternating unitary process should be applied. This expression shows that the QAOA ansatz consists of $2p$ trainable parameters.

Similarly, the **Variational Hamiltonian Ansatz** aims to prepare a trial ground state for a given Hamiltonian $H = \sum_k H_k$ where the terms $\{H_k\}$ do not commute. The difference between this ansatz and QAOA is that in QAOA, only two unitaries (problem unitary and mixer unitary) are alternated. In the Variational Hamiltonian Ansatz, unitaries from all the decomposed terms H_k are used to create the ansatz,

$$U(\boldsymbol{\theta}) = \Pi_l^p (\Pi_k e^{-\theta_{l,k} H_k}) \tag{54}$$

Both of these methods can be viewed as generelized trotterization of adiabatic state evolution. Trotterization means to truncate the the exponential of a sum of two or more hermitian operators, namely $e^{-i(H_1+H_2)t} = \lim_{N\to\infty} \left(e^{-iH_1 t/N} e^{-iH_2 t/N}\right)^N$. Further comparisons between adiabatic state preparation and the QAOA algorithm will be presented in section 5.5.

The QAOA ansatz started as a promising heuristic algorithm for solving combinatorial optimization problems. However, the generalization of the ansatz to different algorithms has proven successful [34]. As a result, this ansatz, in particular, will be further elaborated on in section 5

3.4.3 Expressibility of a circuit

The differences between these types of ansätze can be understood through the notion of *ansatz expressibility*. Ansatz expressibility considers the number of different quantum states reachable by an ansatz if the parameters were to be varied. Holmes et al. [35] quantify this notion by considering the following. Let $\mathbb{U} = \{U^{(1)}, U^{(2)}, \ldots, U^{(y)}\}$ be the corresponding ensemble of unitaries generated by a different set of parameters $\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(y)}\}$. Define $\mathbb{U}_s$ as the set of solution unitaries to a problem (a unitary that is close to the unitary that minimizes the cost). The ansatz is said to be *complete* for a given problem when $\mathbb{U}_s \cap \mathbb{U} \neq \emptyset$. In other words, the ansatz can generate a unitary that solves the problem.

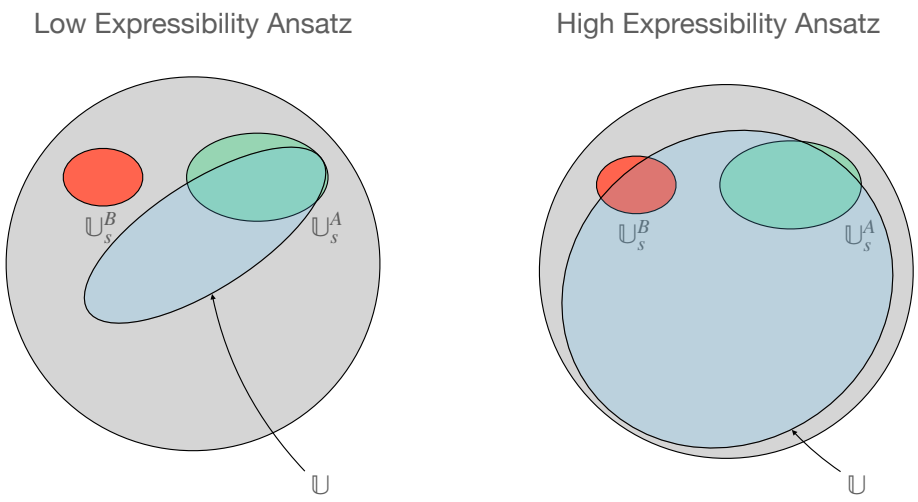


Figure 9: An illustration showing the difference in expressibility between a low- and high-expressible ansatz. $\mathbb{U}$ represents the ensemble of unitaries that can be generated from the ansatz, $\mathbb{U}_s$ represents the solution unitaries to problems A and B, while the grey circle represents the total space of unitaries. The low-expressibility circuit can reach unitaries solving only problem A, while the high-expressibility ansatz is complete for both problems A and B (Adapted from Holmes et al. [35]).

One can distinguish two categories of ansätze in this regard: expressive and inexpressive ansätze. The problem-agnostic ansätze usually refer to expressive ansätze. When there is no information to pinpoint where the optimal unitaries lie, the optimal strategy to create a complete ansatz is for the ansatz to be capable of searching the total space of unitaries. Figure 9 illustrates this notion where the ansatz with high expressibility can find unitaries that solve both problems A and B.

Problem-inspired ansätze on the other hand limits the searchable space of unitaries to find unitaries that solve one problem in particular. For instance, if one were to use the QAOA-ansatz to solve both problems A and B, one would necessarily have to change the problem-hamiltonian H_P in the ansatz to the particular problem being solved. Although these ansätze are complete for the problem to be solved, they are inexpressible as the searchable space of unitaries is restricted. Note that a problem-agnostic ansatz needs to have a significant depth to be sufficiently expressible to be applied to various problems.

The circuit expressibility is the circuit's ability to generate pure states representative of the Hilbert space. In the case of a single qubit, the Hilbert space can be spanned by the general single-qubit unitary

$$U(\phi,\theta,\omega) = \begin{pmatrix} e^{-i(\phi+\omega)/2}\cos(\theta/2) & -e^{i(\phi-\omega)/2}\sin(\theta/2) \\ e^{-i(\phi-\omega)/2}\sin(\theta/2) & e^{i(\phi+\omega)/2}\cos(\theta/2) \end{pmatrix} \tag{55}$$

The expressibility of a single-qubit parametrized circuit is how similar the parametrized circuit unitary and the general single-qubit unitary are to each other. Generalizing this notion to the unitary group $U(N)$ for $N \times N$ unitary matrices can be

done differently depending on how *closeness* is defined. Sim et al. [36] quantify the expressibility of an ansatz $U(\boldsymbol{\theta})$ by comparing the states reachable from this unitary to the ensemble of Haar random matrices. These matrices are uniformly distributed in the space of unitaries measured using the Haar measure $d\mu$.

The Haar measure expresses the distribution of all unitaries in the unitary group $U(N)$. A measure gives information about how the elements of a mathematical set/space are distributed and concentrated. For instance, the sphere is parametrized by three parameters, r, θ, ϕ, and the measure of this space is given by $dV = r^2 \sin(\theta) dr d\theta d\phi$. The measure must be included if a function is integrated over the sphere or points sampled from this space to give the correct expression. The measure extends this notion to abstract mathematical spaces, and the Haar measure, $d\mu$, describes the measure of the unitary group $U(N)$. Every point in the unitary group is a unitary matrix described by certain parameters, similar to how the three coordinates describe the sphere.

Through this measure, one may sample random quantum circuits (unitaries) by sampling random parameter values. From this, circuit expressibility defines how close the circuits generated by a specific ansatz are to the unitary group's unitaries. An estimate of the latter can be made by sampling random circuits in unitary space. To evaluate this analytically, the number of samples needed would technically be infinity. However, using the unitary t-design, one may reduce the sampling procedure only to include a particular set of unitaries.

In these groups, one considers polynomials $P_{t,d}(U)$ with degree at most t in d variables that acts on the elements of a unitary U. The unitary t-design is a set of K unitaries $\{U_k^{(t)}\}$ such that the average over the polynomial $P_{t,d}$ is equal to the average over the Haar measure $d\mu(U)$ of the unitary group [37],

$$\frac{1}{K} \sum_{k=1}^{K} P_{t,d}\left(U_k\right) = \int_{\mathcal{U}(d)} P_{t,d}(U) d\mu(U). \tag{56}$$

This relation should hold for all the unitaries $\{U_k^{(t)}\}$ in the t-design and for all possible polynomials $P_{t,d}$. The above relation is exact. This representative set consists of unitaries evenly spaced in the unitary group $\mathcal{U}(d)$. By only using the unitaries from the t-design, complex functions can be evaluated using this representative set instead of the entire unitary space.

Now, using the difference between the random Haar unitary matrices and the unitaries reachable through the circuit ansatz as a measure of closeness, the expressibility of an ansatz can finally be expressed quantitatively by $||A^{(t)}||$ [36],

$$A^{(t)}(U) := \int dU_{\text{Haar}}\, U_{\text{Haar}}^{\otimes t} \left|0\right\rangle\!\left\langle 0\right| \left(U_{\text{Haar}}^{\dagger}\right)^{\otimes t} - \int dU U^{\otimes t} \left|0\right\rangle\!\left\langle 0\right| \left(U^{\dagger}\right)^{\otimes t}. \tag{57}$$

The first integral represents the space of random Haar unitaries, while the second integral stands for the unitaries spanned by the variational unitary of the circuit.

The closer these integrals are, the larger parts of the unitary group $U(N)$ can be explored by the variational circuit $U(\boldsymbol{\theta})$.

The Haar measure is prone to the concentration of measure, which means that the unitaries tend to cluster in unitary space. Similar concentration is found in the volume of a sphere as there is a higher volume concentration around the equator from the $\sin(\theta)$ term in its measure. This effect is prominent with increasing system sizes. Given a function $f(x)$, Levy's lemma [38] states that on a N-dimensional unit hypersphere, when a point is sampled uniformly at random from the said sphere, the probability that $f(x)$ deviates from the mean of the function $\mathbb{E}[f]$ by an amount ϵ is

$$\Pr(|f(x) - E[f]| \geq \epsilon) \leq 2 \exp\left[-\frac{N\epsilon^2}{9\pi^3\eta^2}\right] \tag{58}$$

given that f is Lipschitz continuous with Lipschitz constant η. In other words, the function evaluation of a random point is exponentially constrained around the mean value, whose effect is exponentially dependent on N, the dimension of the hypersphere. Noting that quantum states are mapped to hyperspheres because $|\langle\psi|\psi\rangle|^2 = 1$, functions on quantum states are also prone to this concentration of measure effect. The consequences of concentration of measure are discussed by Hayden et al. [39]; they find that as the qubit-count increases (the unitaries become larger), the randomly sampled states will concentrate around the maximally entangled state. If an ansatz is highly expressible, the Haar measure of unitary matrices will cause concentration of measure. This concentration ties into a phenomenon called Barren plateaus. Barren plateaus cause trainability issues in circuit ansätze that are highly expressive. This phenomenon is discussed further in section 4.2.

General VQAs can be summarized as follows: One starts with embedding classical information into a quantum state. This state evolves through a set of unitary gates defined by the circuit ansatz that may either be problem-specific or problem-agnostic. Measurements from the final state are the input to a problem-specific cost function. With all these pieces in hand, one can train the variational parameters of the quantum circuits using a classical optimizer. This training procedure is the topic of the next section.

4 Training hybrid models

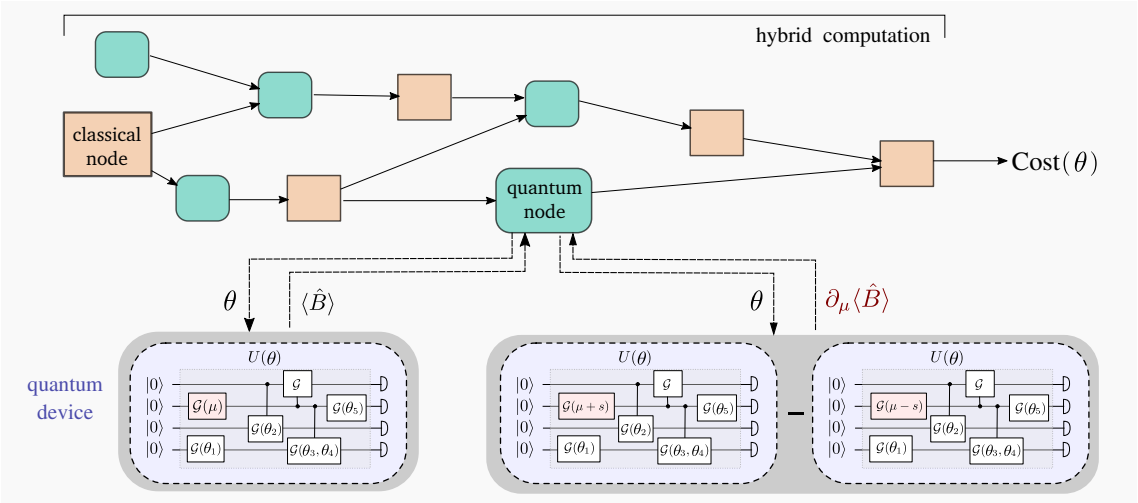


Figure 10: Overview of hybrid computation using both classical and quantum nodes. A diagrammatic representation of the parameter shift rule is also presented, showing the macroscopic shift in parameter to compute the gradient of the quantum node (Schuld 2019 [40]).

A general cost function of classical-quantum hybrid models can contain both classical and quantum information processing nodes, as shown in figure 10. Information is passed from one node to another until the final node calculates the actual cost. Both nodes can have variational parameters that need to be optimized to minimize the cost, often using some gradient-based method.

There are primarily three ways of computing derivatives on a computer: numeric-, symbolic- and auto-differentiation. Numerical differentiation primarily refers to finite difference methods, while symbolic differentiation returns the symbolic expression of the differentiated function using symbolic programming packages.

Automatic differentiation is a method of derivation which takes an arbitrary function expression and reduces it down to its elementary operations (addition, subtraction, multiplication, division) and elementary functions (exp, log, sin). This reduction follows the chain rule. By storing the values and derivatives of the linked interdependencies and reusing them in later calculations, one can calculate the derivative of a function with respect to some arbitrary variable, $\frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i}$. As noted by Baydin et al. [41], "automatic differentiation refers to a set of techniques that are similar, but more general than back-propagation, used to efficiently and accurately evaluate the derivative of numeric functions" (Baydin, 2015).

In order to calculate how the cost changes with respect to the variational parameters within a quantum node, the following chain-rule relation holds

$$\partial_\mu C(\theta) = \frac{\partial C}{\partial Q}\frac{\partial Q}{\partial \mu} \tag{59}$$

where Q represents the quantum node as seen in figure 10 wherein μ is a parameter. By merely defining C and Q, the computational framework can compute the derivatives automatically without further input from the user. The first term represents the change in a classical node with respect to the output results from a quantum node. Classical auto-differentiation libraries like Pytorch and Tensorflow are already capable of differentiating such classical nodes. However, to incorporate differentiable quantum nodes into these auto-differentiation frameworks, rules have to be provided regarding how quantum circuits should be differentiated w.r.t its variational parameters. The auto-differentiation libraries are effective because they can store and reuse intermediate derivatives of an arbitrary functional relation by reducing the expression to the elementary operators and functions. However, storing and reusing intermediate values of the derivative during the quantum computation is impossible for a variational quantum circuit since it requires the measurement of intermediate quantum states, which impacts the overall computation. The rules provided to the auto-differentiation libraries will therefore have to be of a "black-box" type in which changing a parameter results in a new circuit evaluation, from which the derivative can be found. The following section focuses on such a method to differentiate a variational circuit using parameter-shift rules.

4.1 Parameter-shift Rules

The parameter shift rule is an analytic method that calculates the gradients of expectation values of quantum measurements. Similar to central finite differences, this method measures the same circuit (model function) twice, shifting the parameter of a single gate, as diagrammatically shown in the second part of figure 10. The significant difference between these methods is how large this shift is, as will be discussed later in this section. The parameter shift rule, presented here, follows the derivation presented by Mari et al. [42].

A variational quantum model is usually expressed as the expectation value of some Hermitian observable M

$$f(\boldsymbol{\theta}) = \left\langle 0 \left| U(\boldsymbol{\theta})^\dagger M U(\boldsymbol{\theta}) \right| 0 \right\rangle \tag{60}$$

where $U(\boldsymbol{\theta})$ is a set of unitary operators that depend on classical variational parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_m)$, whose gradients need to be found. Generally, the unitary can be written as

$$U(\boldsymbol{\theta}) = V_m U_m(\theta_m) \ldots V_2 U_2(\theta) V_1 U_1(\theta_1) \tag{61}$$

where V_j are parameter-independent circuits while $U_j(\theta_j)$ are characterized by involutory matrices H_j, i.e matrices where $H_j^2 = \mathbb{1}$. Examples of such generator-matrices are the single and multi-qubit Pauli rotation gates. Unitary gates involving involutory matrices can, in general, be written as

$$U_j\left(\theta_j\right) = e^{-\frac{i}{2}H_j\theta_j} = \cos\left(\theta_j/2\right)\mathbb{1} - i\sin\left(\theta_j/2\right)H_j. \tag{62}$$

As is often the case with VQAs, the expectation of an operator is measured. Consider, therefore, the unitary conjugation of an arbitrary operator $\hat{K}$ by $U_j(\theta_j)$.

$$\hat{K}(\theta_j) = U_j\left(\theta_j\right)^\dagger \hat{K} U_j\left(\theta_j\right) = \big(\cos(\theta_j/2)\mathbb{1} + i\sin(\theta_j/2)H_j^\dagger\big)\hat{K}\big(\cos(\theta_j/2)\mathbb{1} - i\sin(\theta_j/2)H_j\big) \tag{63}$$

$$= \hat{A} + \hat{B}\cos(\theta_j) + \hat{C}\sin(\theta_j) \tag{64}$$

where $\hat{A}, \hat{B}, \hat{C}$ are only dependent on $\hat{K}, \hat{H}_j$ and thus independent of the parameters to be differentiated. Standard trigonometric addition and subtraction identities can be expressed as follows

$$\frac{d\cos(x)}{dx} = \frac{\cos(x+s) - \cos(x-s)}{2\sin(s)} \tag{65}$$

$$\frac{d\sin(x)}{dx} = \frac{\sin(x+s) - \sin(x-s)}{2\sin(s)}. \tag{66}$$

This identity is valid for any shift $s \neq k\pi, k \in \mathbb{Z}$. By using this identity, $\hat{K}$ can be differentiated with respect to θ_j, yielding

$$\frac{\partial}{\partial\theta_j}\hat{K}(\theta_j) = \frac{\cos(x+s) - \cos(x-s)}{2\sin(s)}\hat{B} + \frac{\sin(x+s) - \sin(x-s)}{2\sin(s)}\hat{C}. \tag{67}$$

Adding and subtracting the above equation with $\hat{A}$ gives an illuminating expression for the derivative of the unitary conjugated operator,

$$\frac{\partial}{\partial\theta_j}\hat{K}\left(\theta_j\right) = \frac{\hat{K}\left(\theta_j + s\right) - \hat{K}\left(\theta_j - s\right)}{2\sin(s)}. \tag{68}$$

Since no assumptions on the operator K were made, this expression can be applied to the conjugated operator in the original cost-function $f(\boldsymbol{\theta})$. The parameter-shift rule to evaluate the j_{th} component of the gradient is given as

$$g_j(\boldsymbol{\theta}) = \frac{f\left(\boldsymbol{\theta} + s\mathbf{e}_j\right) - f\left(\boldsymbol{\theta} - s\mathbf{e}_j\right)}{2\sin(s)} \tag{69}$$

where $\mathbf{e}_j$ is the unit vector in the j-th direction. This expression expresses that in order to calculate the j_{th} component of the gradient, one has to perform two circuit measurements with the parameter θ_j in the unitary $U_j(\theta_j)$ shifted by $\pm s$ while keeping all other parameters unshifted. At first glance, this expression looks similar to central finite differences

$$\frac{\partial f(\theta)}{\partial \theta} \approx \frac{f(\theta + \Delta\theta) - f(\theta - \Delta\theta)}{2\Delta\theta}. \tag{70}$$

Specifically, the $s \to 0$ limit gives the central finite difference approximation since $\sin(s) \approx s$ in this limit. However, there are notable differences. Most notably, the gradient from the parameter-shift rule is exact, not approximated as in finite differences. Naturally, this expectation value can only be estimated on hardware through many calls to the quantum computer. The differences between the two are therefore evident; parameter shifting allows for the computation of an estimate of the analytic gradient, whereas finite differences estimate the approximate gradient.

Since finite differences linearly approximate the gradient at a point, smaller values of $\Delta\theta$ yield better approximations of the gradient (in practice, this is true until rounding errors become notable). This is not the case for the parameter shift rule since the formula is exact for arbitrary s except multiples of π, which is why parameter-shift is preferable over finite differences. If the shift $\Delta\theta$ in finite differences is too small ($\Delta\theta \ll 1$), the shifted function evaluations cannot be differentiated from noise. In contrast, a macroscopic shift s in parameter space allows for function evaluations that are less likely to overlap. Although both methods are prone to noise, the parameter-shift method is not prone to numerical issues. A macroscopic shift of $s = \pi/2$ is often used in libraries like Pennylane [43].

Applying the parameter shift rule twice yields an analytical expression for the Hessian. When applying the parameter shift rule twice, one can, in principle, use two different shifts s_1, s_2; however, both are commonly set to the same value s. Using this simplification, the Hessian is expressed as

$$g_{j_1,j_2}(\boldsymbol{\theta}) = \frac{1}{4\sin^2(s)}\Big[\, f\left(\boldsymbol{\theta} + s\left(\mathbf{e}_{j_1} + \mathbf{e}_{j_2}\right)\right) - f\left(\boldsymbol{\theta} + s\left(-\mathbf{e}_{j_1} + \mathbf{e}_{j_2}\right)\right) \tag{71}$$

$$- f\left(\boldsymbol{\theta} + s\left(\mathbf{e}_{j_1} - \mathbf{e}_{j_2}\right)\right) + f\left(\boldsymbol{\theta} - s\left(\mathbf{e}_{j_1} + \mathbf{e}_{j_2}\right)\right) \,\Big] \tag{72}$$

As with the gradient, the $s \to 0$ limit gives the finite differences approximation of the Hessian.

As mentioned earlier, there are certain limitations on the applicability of this specific parameter shift rule, namely that the generator matrices H needed to be involutory. However, the stochastic parameter shift rule, introduced by Banchi et al. [44], can address parameter shifts for an arbitrary generator matrix by first expanding the generator into a sum of Pauli-Strings,

$$\hat{G} = \hat{A} + \theta_i \hat{V} \tag{73}$$

where $\hat{A}$ is an arbitrary linear combination of Pauli strings while $\hat{V}$ is a single arbitrary Pauli string, this is excluded from the rest precisely due to the variational parameter θ_i, which is the variable to be differentiated. The algorithm computing the derivative of the expectation of an observable $\langle M(\theta) \rangle$ is the following:

1. Choose a value s uniformly from [0,1]
2. Rather than applying $e^{i\hat{G}}$, apply instead $e^{i(1-s)(\hat{A}+\theta_i\hat{V})}e^{i\frac{\pi}{4}\hat{V}}e^{is(\hat{H}+\theta_i\hat{V})}$

   Perform repeated measurements of M and call the expectation value $\langle r_+ \rangle$
3. Repeat the previous step, but flip the angle of $\frac{\pi}{4}$ to $-\frac{\pi}{4}$ in the second gate. Call this measurement expectation value $\langle r_- \rangle$

   The gradient is then given by $\mathbb{E}_{s\in\mathcal{U}[0,1]}\left[\langle r_+ \rangle - \langle r_- \rangle\right]$

General parameter shift methods can be made for more general gates without restricting to only two eigenvalues in the generator G. Wierichs et al. [45] looked into this problem. Their approach hinges on the notion that the expectation value of an observable with respect to a unitary evolution $U(x) = \exp(ixG)$ with free parameter x may be decomposed as a Fourier series

$$E(x) := \left\langle \psi \left| U^\dagger(x) M U(x) \right| \psi \right\rangle \tag{74}$$

$$= \sum_{j,k=1}^{d} \overline{\psi_j e^{i\omega_j x}} m_{jk} \psi_k e^{i\omega_k x} \tag{75}$$

$$= \sum_{\substack{j,k=1 \\ j<k}}^{d} \left[\overline{\psi_j} m_{jk} \psi_k e^{i(\omega_k-\omega_j)x} + \psi_j \overline{m_{jk}\psi_k e^{i(\omega_k-\omega_j)x}}\right] + \sum_{j=1}^{d} |\psi_j|^2 m_{jj}, \tag{76}$$

$$= a_0 + \sum_{\ell=1}^{R} a_\ell \cos(\Omega_\ell x) + b_\ell \sin(\Omega_\ell x). \tag{77}$$

In the first two equations, M and $|\psi\rangle$ was expanded in the eigenbasis of U as denoted by m_{jk} and ψ_j. Note also that the eigenvalues of $U(x) = \exp(ixG)$ are $e^{i\omega_j x}$ with real valued ω_j and $j \in \{1, \ldots, d\}$ where d is the dimension of the generator G. In the last equality, R was introduced as the number of unique positive differences $\{\Omega_l\} = \{\omega_k - \omega_j | \omega_k > \omega_j\}$ with $j, k = \{1, \ldots, d\}$ and $l = \{1, \ldots, R\}$.

The important observation from the above decomposition is that equation 77 shows that the expectation value E can be decomposed into a finite-term Fourier series. Therefore, if one were to have 2R + 1 interpolating points $\{x_\mu\}$, one could completely determine the unknown coefficients $\{a_l, b_l\}$ using a nonuniform discrete Fourier transform. $E(x)$ is then constructed and using it one can differentiate it for

arbitrary x. If more assumptions are made, namely that the spectra $\Omega_l = l\Omega$ and that $\Omega = 1$, one could get a closed form expression of $E(x)$ by evaluating them at points $\left\{x_\mu = \frac{2\mu}{2R+1}\pi\right\}, \mu \in \{-R, \cdots, R\}$

$$E(x) = \sum_{\mu=-R}^{R} E\left(x_\mu\right) D\left(x - x_\mu\right) = \frac{\sin\left(\frac{2R+1}{2}x\right)}{2R+1} \sum_{\mu=-R}^{R} E\left(x_\mu\right) \frac{(-1)^\mu}{\sin\left(\frac{x-x_\mu}{2}\right)} \tag{78}$$

where $D(x - x_\mu)$ are Dirichlet kernels, often used when interpolating trigonometric functions. The most important takeaway is that the entire function $E(x)$ can be reconstructed using $2R+1$ function evaluations of $E(x_\mu)$, constructed through calls to the quantum computer.

This subsection concludes with how one step of straightforward gradient descent scales when using the parameter shift rule to evaluate the gradient. When using the simple two-term parameter shift rule considered earlier, the number of circuit evaluations scales as $2JS$ where J is the number of parameters in the circuit and S is the number of shots used to estimate a single expectation value. In terms of scaling, gradient-calculations scale worse with increased parameters J. This is because each of the J parameters' gradients needs to be calculated separately in contrast to traditional auto-differentiation methods such as backpropagation, whereby storing intermediate values of the derivatives, the total gradient can be calculated using a single evaluation of the model. Implementations similar to those in auto-differentiation libraries can only be implemented in simulators of quantum computers. Using backpropagation-like methods through a quantum circuit would prove impossible since storing the states being differentiated is prohibited by the no-cloning theorem.

For the general parameter shift rule, the scaling is worse and goes like $(2R+1)JS$ since each parameter x requires the reconstruction of $E(x)$ to be differentiated. Wierichs et al. [45] also provide closed-form solutions of $E'(0)$; however, the scaling remains the same. R was the number of unique positive differences Ω_l needed to recompose $E(x)$ which has a scaling of $R \leq \frac{r(r-1)}{2}$ with $r = |\{\omega_j\}|$ being the number of unique eigenvalues of the generator G. As evident from these scaling arguments, the parameter shift rule is a costly way to calculate gradients, at least comparatively to finite differences, which would only require $2JS$ function evaluations for the entire gradient for an arbitrary quantum function. However, as mentioned earlier, this comes with associated numerical instability, which causes infeasible function comparisons on NISQ hardware.

4.2 Barren Plateaus

Although the parameter shift rule does allow for the navigation of the cost landscape (i.e., regimes for the parameters $\boldsymbol{\theta}$), two problematic landscapes may occur during training. The first type of landscape has several suboptimal local minima (discussed in the following subsection), while the other is essentially flat. The latter type categorizes Barren Plateaus, landscapes with gradient elements close to zero. Such

landscapes cause slow optimization since there is no reliable information on the direction for the steepest gradient. Additionally, small gradients are difficult to differentiate from noise, which requires high precision measurements to avoid random walks in the cost landscape.

Barren Plateaus have been observed in variational circuit architectures that are very expressive (i.e., capable of exploring a large Hilbert space). Since the number of parameters, in this case, is large, the contribution from a single parameter becomes negligible, giving an intuitive explanation as to why Barren Plateaus are prevalent in these circuits. This result contrasts with traditional machine learning models, where more parameters yield increased trainability. Most works on this topic utilize randomized circuits without any particular structure to both numerically and analytically show that the gradient vanishes. The Haar-Measure is used to evaluate these circuits analytically.

Assuming a generic ansatz of form $U(\boldsymbol{\theta}) = \prod_{j=1}^{D} U_j(\theta_j) W_j$ with $U_j = e^{-i\theta_j V_j}$ and hermitian operator V_j that satisfies $(V_j)^2 = \mathbb{1}$, Holmes et al. [35] show two notable aspects about the gradient of the cost landscape when certain assumptions are made.

Firstly, the authors note that

$$\langle \partial_k C \rangle = 0 \;\forall\; k, \quad \partial_k C := \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_k} \tag{79}$$

The result is found through explicit calculations by integration over θ_k. Intuitively, this unbiased result can be understood by noting that the average of a rotation unitary $e^{-i\theta_k V_k}$ is zero when $V_k^2 = \mathbb{1}$. Fluctuations away from zero mean value allow for training since the gradient has a finite direction. The probability of such fluctuations can be bounded using Chebyshev's inequality

$$P(|\partial_k C| \geq \delta) \leq \frac{\mathrm{Var}[\partial_k C]}{\delta^2} \tag{80}$$

$$\mathrm{Var}[\partial_k C] = \left\langle (\partial_k C)^2 \right\rangle - \langle \partial_k C \rangle^2. \tag{81}$$

where the expectation value is taken over the parameters $\boldsymbol{\theta}$. As evident by Chebyshev's inequality, if the variance of the partial derivative $\partial_k C$ vanishes exponentially for all directions θ_k, the probability of a non-zero $\partial_k C$ becomes exponentially small, resulting in an inability to navigate the cost landscape since the steps taken are exponentially small. Therefore, the dependency on the variance of the cost derivative is crucial, where a low variance results in slow or even inability to train.

The barren plateau phenomenon is a probabilistic result that states that given an ansatz $U(\boldsymbol{\theta})$, the cost has a barren plateau if the gradient vanishes exponentially with the number of qubits n with a high probability. As mentioned above, the variance of the cost-derivative captures this notion. To show that the variance does vanish exponentially, one starts by separating the ansatz into two

$$U(\boldsymbol{\theta}) = U_L(\boldsymbol{\theta})U_R(\boldsymbol{\theta}) \tag{82}$$

$$U_L(\boldsymbol{\theta}) = \prod_{j=k+1}^{D} U_j\left(\boldsymbol{\theta_j}\right) W_j \quad \text{and} \quad U_R(\boldsymbol{\theta}) = \prod_{j=1}^{k} U_j\left(\boldsymbol{\theta}_j\right) W_j \tag{83}$$

where the right unitary contains the parameter θ_k to be differentiated. The parameters θ_j are assumed uncorrelated; therefore, these unitaries are independent. A critical assumption is that the ensemble of unitaries $\mathbb{U}_L$ and $\mathbb{U}_R$ formed by these two unitaries form a 2-design. Unitary 2-designs are particular instances of the general group of unitary t-designs as elaborated on when discussing circuit expressibility. The assumption of unitary 2-designs in this derivation holds for a widely studied hardware-based expressive ansatz built up in a brick-wall-like manner using 2-qubit gates acting layer-wise on alternating pairs of qubits, studied by Cerezo et al. [46]. The result from these calculations is that if either or both of the $\mathbb{U}_L, \mathbb{U}_R$ ensembles form a 2-design, the qubit-dependence of the variance takes the following form:

$$\mathrm{Var}_x\, \partial_k C = \frac{g_x(\rho, H, U)}{2^{2n} - 1} \tag{84}$$

Here, $\mathrm{Var}_x \partial_k C$ refers to the variance of the cost gradient when either R, L or both unitaries are 2-design. This proof is based on using that 2-design unitaries have expectations that up to second-order behaves like the Haar Distributions. Applying some identities for such distributions and calculating yields the answer. The function g_x depends on the problem being solved.

This result generally holds for deep, expressive circuits capable of being applied to multiple problems. The phenomenon has also been shown to appear in shallow circuits [46], where local instead of global cost functions have been shown to have better trainability. Additionally, Want et al. [47] rigorously prove that noise can also induce barren plateaus.

From these considerations, it seems that less expressive circuits that limit the search to relevant parts of Hilbert space are preferable over expressible unitaries. In this regard, few studies have been performed on the analytical side of barren plateaus for hamiltonian-inspired circuits. In a recent contribution to the topic, Wiersema et al. [48] show numerical evidence that when solving a particular problem using the variational Hamiltonian ansatz, only small barren plateaus are encountered. The authors also encountered a phase transition in the local minima distribution, potentially aiding the training of VQA.

4.3 Local minima distribution

A natural issue related to quantum circuits' trainability is local minima distribution. The cost landscape of VQAs is widely recognized as non-convex and therefore filled

with several sub-optimal local minima. This is also the case with traditional machine learning models. In high-dimensional neural networks, it is found that when using random Gaussian landscapes as a way to approximate the error landscape of random neural networks, some interesting properties emerge. Most notably, critical points higher in energy primarily are saddle points, and the probability of encountering a local minimum in which all directions curve upwards is exponentially small [49]. As a result, traditional neural networks cluster local minima around the energy of the global minima. When studying high-dimensional neural network error landscapes, the Random Gaussian Landscape approximation is reasonable [50]. This methodology stems from random matrix theory and is often used to analyze neural networks' error landscapes.

In contrast, mapping a random VQA to random Gaussian fields is impossible because of two factors: the first is that the Gaussian mapping relies on the non-linearities of the learning model, which are absent in VQAs since the state evolution is unitary. Secondly, compared to traditional learning models, variational circuits are heavily under-parameterized since variational unitaries consist of a single variational parameter for a potentially exponentially sized matrix. In contrast, every single matrix element in neural networks can be trained.

Anschuetz [11] studied randomized variational hamiltonian ansätze of the form $\Pi_i^q e^{-i\theta_i Q}|\psi_0\rangle$ where Q is a uniformly random Pauli matrix. He finds that due to the limitations mentioned above of VQA and the random nature of the ansatz itself (because of the random sampling of Q in the ansatz), the mapping of error landscapes to random Gaussian fields is not valid. Instead, he finds that with the assumptions made for this particularly randomized ansatz, mapping to the Wishart random fields on a hypertorus is appropriate in contrast to the Gaussian hypersphere,

$$\text{Gaussian Mapping: } F_{\text{GHRF}}(\boldsymbol{\theta}) \propto \sum_{i_1,\ldots,i_r,i'_1,\ldots,i'_r=1}^{\Lambda} \sigma_{i_1}\ldots\sigma_{i_r} J_{i_1,\ldots,i_r,i'_1,\ldots,i'_r}\sigma_{i'_1}\ldots\sigma_{i'_r} \quad (85)$$

$$\text{Wishart Mapping: } F_{\text{WHRF}}(\boldsymbol{\theta}) \propto \sum_{i_1,\ldots,i_r,i'_1,\ldots,i'_r=1}^{2^p} w_{i_1}\ldots w_{i_r} J_{i_1,\ldots,i_r,i'_1,\ldots,i'_r} w_{i'_1}\ldots w_{i'_r} \quad (86)$$

where $\boldsymbol{w}$ are points on a hypertorus while $\boldsymbol{\sigma}$ are points on a hypersphere, both parametrized by $\boldsymbol{\theta}$ (the geometry of these fields come from the 2π periodicity in VQA because the parameters are angles, while such limitations are not present in neural network parameters). p stands for the number of distinct parameters θ_i while r stands for the fraction of unitaries that are distinct, $r = q/p$. The difference is that each effective interaction $\boldsymbol{J}$ is a $r \times r$ random Gaussian matrix. In contrast, the interaction in the Wishart case, $\boldsymbol{J}$ is a $r \times r$ complex random Wishart matrix normalized by its degree of freedom, which scales like $m = \mathcal{O}(2^n)$ where n is the number of qubits. Wishart matrices are a generalization of gamma distribution for multiple dimensions.

This mapping shows that the local minima distribution exhibits a phase transition

when the number of parameters exceeds a threshold that depends exponentially on the number of qubits in the system. The findings culminate into the definition of an order-parameter $\gamma = \frac{p}{2m}$. When $\gamma \leq 1$, the model is under-parameterized, the local minima distribution is exponentially centered around half the mean eigenvalue of the observable O of the cost function. Once the number of distinct parameters p exceeds m, the variational circuit is over-parametrized, and the local minima are exponentially centered around the global minima. As mentioned, the problem with this transition is that m scales exponentially in the number of qubits.

This computational phase transition is also found numerically. Kiani et al. [12] studied the task of learning an arbitrary unitary $d \times d, d = 2^N$ matrix using the QAOA ansatz. They define learning as having the QAOA circuit transformation approximate this arbitrary unitary. The problem and mixer Hamiltonians used in their setup were two matrices sampled from the Gaussian Unitary Ensemble. When the number of $2p$ parameters that define the QAOA ansatz is less than d^2, the gradient descent optimization converged to a suboptimal solution. However, when $2p > d^2$, the optimization *always* converged to the global minimum, exhibiting the computational phase transition shown by Anschuetz [11]. The rate of convergence follows a power-law in the under-parametrized region, however transitions to an exponential convergence in the over-parametrized region. The rate of convergence is for the steps in gradient descent. At the critical point $2p = d^2$, the authors find a power-law convergence to the global minimum.

These two subsections paint a grim picture for randomly initialized problem-agnostic ansätze, where the training results in suboptimal solutions for circuits of low depth and training becomes infeasible due to Barren plateaus at high depths. Such rigorously detailed analytical analyses have yet to be performed on problem-inspired ansätze. However, as mentioned earlier, Wiersema et al. [48] show that when applying the Hamiltonian Variational Ansatz on two different 1D Ising chain-type Hamiltonians, the same computational phase transition from under-parametrized to over-parametrized ansätze was found. Most strikingly is the discovery that this phase-transition does not scale exponentially in the problem size but instead scales at most polynomially with problem size. This phase transition is also different from the one seen in learning arbitrary unitaries using hamiltonian agnostic circuits from Kiani et al. [12] in the sense that this depth threshold is not tight. This means that even for lower depth, several random initializations were able to use gradient descent to converge to a good solution.

In principle, this over-parametrized regime improves the quantum model's trainability by concentrating the cost function's local minima close to the global minima. However, this implies that the circuit needs to reach a certain depth, which requires longer coherence times from the qubits and adds multiplicative gate errors. This depth, with its parameters, needs to have its gradient calculated using the parameter shift rule, which scales linearly with the number of qubits. Hence training becomes slower (at least when compared to differentiation of neural network, which requires a forward and backward pass of auto-differentiation). As a result, researchers are not sure whether the over-parametrized regime is attainable on near-term devices.

The parameter-shift rule allows for the training of variational quantum circuits

within an overall hybrid classical-quantum model using classical optimization. As alluded to in this section, training the quantum nodes may prove challenging, mainly because of non-convex cost landscapes causing sub-optimal solutions to be found and the emergence of barren plateaus as the number of qubits increases. The following section considers a particular VQA called QAOA in detail to put these concepts into practice. The motivation behind studying this particular VQA is the suggestion that this algorithm might be promising in showing quantum advantage [51].

5 Quantum Approximate Optimization Algorithm (QAOA)

The Quantum Approximate Optimization Algorithm (QAOA) is a VQA designed to solve combinatorial optimization problems. These problems are defined on N-bit binary strings $z = \{0,1\}^N$ with the goal of finding the string that maximizes a cost function $C(\boldsymbol{z}) : \{0,1\}^N \to \mathbb{R}$. An approximate optimization algorithm aims to find a string z which can approximate the optimal solution within a desired approximation ratio

$$\frac{C(z)}{C_{max}} \geq r \tag{87}$$

$$C_{max} = \max_z C(z) \tag{88}$$

In order to solve a classical combinatorial problem on a quantum system, one needs to convert the classical cost function into a problem-Hamiltonian which encodes the solution to the classical problem. Such encoding is constructed by mapping the binary values z_i onto the eigenvalues of the Pauli Z operator:

$$H_C = C(\sigma_1^z, \sigma_2^z, \ldots, \sigma_N^z), \quad \sigma_i^z = \{-1, 1\} \tag{89}$$

The QAOA ansatz starts in the uniform superposition $|+\rangle^N$ and alternately applies the problem Hamiltonian H_C and a mixer Hamiltonian H_B for short times γ_i and β_i respectively. The most crucial aspect of the mixer Hamiltonian is that it does not commute with the problem Hamiltonian. The non-commutability between H_C and H_B is crucial since it increases the expressibility of the ansatz as more states are reachable. Without the mixer, the ansatz would get stuck in a local optimum of the problem Hamiltonian. To see why, consider the ansatz with only the problem unitary, $e^{-i\gamma H_P}$. If a state, $|\psi\rangle$, prior to applying the problem unitary is an eigenstate of H_P, then the state after the measurement is

$$|\psi(\gamma)\rangle = e^{-i\gamma H_P}|\psi\rangle = \prod_{n=1}^{\infty} \frac{(-i\gamma)^n}{n!} H_P^n |\psi\rangle = \prod_{n=1}^{\infty} \frac{(-i\gamma)^n}{n!} E^n |\psi\rangle = e^{-i\gamma E}|\psi\rangle \tag{90}$$

where E is the energy corresponding to the state $|\psi\rangle$, which is an eigenstate of H_P, but not necessarily the ground state of interest. The mean value of the Hamiltonian with respect to this evolved state, $\langle\psi(\gamma)|H_C|\psi(\gamma)\rangle$, will therefore be the same as for the state prior to the evolution. This mean value is fed into the classical optimizer to be optimized. However, since the variational parameter does not alter this mean value, it cannot be lowered further. Hence, the optimizer remains trapped in a suboptimal eigenstate of the problem Hamiltonian H_P. The same arguments apply

to a mixer Hamiltonian that commutes with H_P. A popular choice for the mixer Hamiltonian is $H_B = \sum_{i=1}^{N} X_i$ (sum of Pauli X on all qubits). Repeating the alternating unitaries p times prepares the parametrized state

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = e^{-i\beta_p H_B} e^{-i\gamma_p H_C} \dots e^{-i\beta_1 H_B} e^{-i\gamma_1 H_C} |+\rangle^{\otimes N}. \tag{91}$$

The $2p$ variational parameters γ_i, β_i $(i = 1, 2, \dots, p)$ describe the applied duration of these unitaries. Similarly to other VQAs, the goal of QAOA is to find the optimal set of parameters $(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)$ that maximizes the expectation value of the cost function:

$$(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*) = \arg\max_{\boldsymbol{\gamma}, \boldsymbol{\beta}} \langle \psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) | H_C | \psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) \rangle \,. \tag{92}$$

In the same way as with classical approximate optimization algorithms, the performance of the QAOA is estimated using the approximation ratio

$$r = \frac{\langle \psi_p(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*) | H_C | \psi_p(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*) \rangle}{C_{max}}. \tag{93}$$

Performance guarantees of the QAOA ansatz have been widely studied for the $p = 1$ case [9], however little is known beyond this depth. Fahri et al. [51] claims that, under certain complexity-theoretic assumptions, sampling from the variational wave function $|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$ is classically intractable. This is the case for the lowest depth QAOA circuit, i.e., $p = 1$. Therefore, the QAOA ansatz is believed to be promising to prove a quantum advantage over classical models.

5.1 QAOA on the MaxCut problem

One combinatorial optimization problem often considered when using QAOA is the MaxCut problem. Given a graph $G = (V, E)$ with V nodes and E edges, divide the graph into two sets such that the number of edges between the two sets of nodes is maximized. It is proven that finding an approximate solution to MaxCut beyond $16/17 \approx 0.9412$ is NP-Hard [52, 53]. The polynomial-time approximation algorithm, the Goemans-Williamson algorithm, guarantees an approximate ratio of 0.8785 [54]. Assuming the unique games conjecture (a specific conjecture from computational complexity theory) to be true [55], this approximation ratio is proven to be optimal [56].

The cost function of MaxCut on a graph G with a set of edges E and nodes V is given as

$$\text{MaxCut}(G) = \max \frac{1}{2} \sum_{i,j \in E} w_{ij}(1 - x_i \cdot x_j), \quad \text{s.t } x_i \in \{-1, 1\} \text{ for every } i \in V \tag{94}$$

where w_{ij} is finite if there is an edge between nodes i, j and 0 otherwise. This cost function gives the expected behavior of MaxCut since when two nodes are of opposite type ($x_i = 1, x_j = -1$), the edge's cost contribution is one. In comparison, if two nodes have the same value and are within the same partition, the cost contribution is zero. Replacing the x_i, x_j variables with the Pauli Z operators encode the classical cost function into a problem Hamiltonian, which is diagonal in the computational basis:

$$H_P = \frac{1}{2} \sum_{i,j \in E} w_{ij}(1 - Z_i Z_j) \tag{95}$$

Z is the Pauli σ^z matrix applied on qubit i. Notice that the behavior of this problem is identical to the classical case since the eigenvalue of the Pauli Z matrix is $\{-1, 1\}$. As evident by the Hamiltonian, this is essentially an energy minimization problem for an anti-ferromagnetic system with coupling w_{ij}. In order to see an implementation of QAOA, consider first a simple graph instance as shown in figure 11. The MaxCut on this graph has a cost equalling 4, where all cuts that separate the nodes into subgroups of two nodes give the maximum cut.

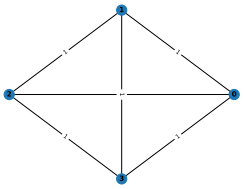
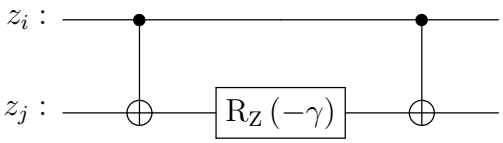


Figure 11: A 3-degree graph with 4 nodes.

To implement QAOA, two operators are needed, namely $e^{-i\gamma H_P}$ and $e^{-i\beta H_B}$. Consider first $e^{-i\gamma H_P} = e^{-i\gamma \frac{1}{2} \sum_{ij \in E}(1 - Z_i Z_j)}$ where i, j are all the nodes with edges on the graph. The operator needed to be considered is $e^{i\frac{\gamma}{2} ZZ} = e^{i\frac{\gamma}{2} Z \otimes Z}$ since the identity term contributes an unobservable global phase. Since the problem is basis-encoded in the computational basis, $e^{-Z}|z\rangle = e^{(-1)^z}|z\rangle$ where $z \in \{0, 1\}$ with eigenvalues $\{1, -1\}$ respectively. The following identity therefore holds for the 2-qubit operator: $e^{i\frac{\gamma}{2} Z \otimes Z}|z_1 z_2\rangle = e^{i\frac{\gamma}{2}(-1)^{z_1 \oplus z_2}}|z_1 z_2\rangle$ where $\oplus$ is mod 2 addition. The quantum circuit that implements this is the following circuit:

$$e^{i\frac{\gamma}{2} Z_i \otimes Z_j} \equiv$$



To show that this circuit indeed does transform the state as stated previously, one can view the quantum state after each gate transformation as follows

$$\mathrm{CNOT}\Big(|z_1\rangle|z_2\rangle\Big) = |z_1\rangle|z_1 \oplus z_2\rangle \tag{96}$$

$$\Big(\mathbb{1} \otimes e^{i\frac{\gamma}{2} Z}\Big)|z_1\rangle|z_1 \oplus z_2\rangle = |z_1\rangle e^{i\frac{\gamma}{2}(-1)^{z_1 \oplus z_2}}|z_1 \oplus z_2\rangle \tag{97}$$

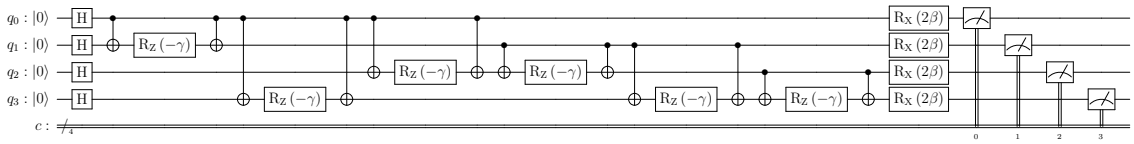
$$e^{i\frac{\gamma}{2}(-1)^{z_1 \oplus z_2}} \mathrm{CNOT}\Big(|z_1\rangle|z_1 \oplus z_2\rangle\Big) = e^{i\frac{\gamma}{2}(-1)^{z_1 \oplus z_2}}|z_1\rangle|z_2\rangle \tag{98}$$

Note also that the terms in the problem Hamiltonian commutes since all terms are either the Z or the I operators, ie $e^{i\frac{\gamma}{2}Z_iZ_j+i\frac{\gamma}{2}Z_jZ_k} = e^{i\frac{\gamma}{2}Z_iZ_j}e^{i\frac{\gamma}{2}Z_jZ_k}$. Implementing the entire $e^{-i\frac{\gamma}{2}H_P}$ operator therefore consists of implementing the above circuit for all edge-combinations on the graph.

The mixer operator $e^{-i\beta H_B} = e^{-i\beta\sum_{i\in V}X_i}$ consists of a block of $R_X(\beta)$ rotations

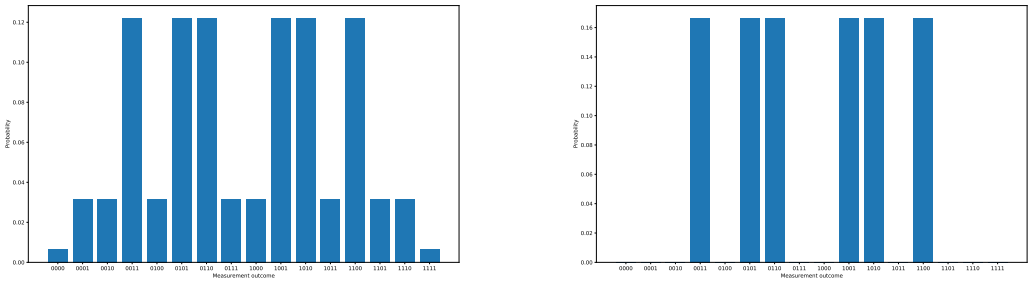
$$e^{-i\beta X_i} \equiv \text{---}\boxed{R_X(2\beta)}\text{---}$$

As noted earlier in the section, the initial state of the QAOA algorithm is in the $|+++\cdots+\rangle$ state, which can be readily prepared using a layer of Hadamard gates. The QAOA algorithm with depth $p = 1$ is given by the unitary $U(\gamma, \beta) = e^{-i\gamma H_P}e^{-i\beta H_B}$, which can be implemented as follows



Repeating this circuit (except the first layer of Hadamard gates and final measurements) p times with different parameters γ_i, β_i gives the QAOA circuit for more considerable depths. After optimizing the parameters, running the above circuit and performing measurements is equivalent to sampling from the distributions shown in figure 12. As evident from the figure, running the algorithm for higher depths results in measurements involving the correct cut with higher probability. For this graph instance, the set of bitstrings $\{0011, 0101, 0110\}$ and their bit-flipped counterparts all give a cost of 4 and solve the MaxCut problem.

Here it should be noted that these results were obtained using a classical statevector simulator, which uses a matrix representation of the circuit to evaluate it precisely. For instance, evaluating the cost function of the Hamiltonian is done by



(a) Circuit depth $p = 1$ (b) Circuit depth $p = 5$

Figure 12: The probability distribution for measuring different basis states using QAOA. These results were obtained using a quantum state-vector simulator.

$$F(\boldsymbol{\gamma}, \boldsymbol{\beta}) = = \langle \psi(\boldsymbol{\gamma}, \boldsymbol{\beta}) | H_C | \psi(\boldsymbol{\gamma}, \boldsymbol{\beta}) \rangle \tag{99}$$

$$= \sum_i \lambda_i p_i \tag{100}$$

where λ_i is an eigenvalue of the operator and p_i is the corresponding probability of measuring this eigenvalue. On physical hardware, one would necessarily have to perform shots to the quantum computer to approximate the probability distribution p_i, which is not the case on a statevector simulator.

The seminal paper on QAOA introduced by Fahri et al. [9] examines the performance of QAOA using the $p = 1$ depth analytically, and the authors find that the algorithm creates a guaranteed approximation rate $r = 0.6924$ on 3-regular unweighted graphs. These are graphs where each node has three other nodes connected, and their weights are 1. By noting that the expectation value (eq. 99) is a sum of all possible subgraphs that can be generated at a given p, Fahri et al. [9] has analyzed the QAOA algorithm for $p = 1$. However, little is understood of QAOA as the depth increases. A critical aspect of solving the MaxCut problem is to optimize the $2p$ QAOA parameters using classical optimization routines using either gradient-based or gradient-free methods. The classical optimization part is prone to encounter sub-optimal solutions due to the non-convex cost landscape, even at low circuit depths. As a result, the performance of the QAOA algorithm is heavily dependent on the initial parameters. Standard approaches that rely on random initialization of parameters quickly become intractable due to a large amount of sub-optimal solutions (unless the depth is large enough to undergo a phase transition as discussed earlier).

A straightforward way to reduce the number of initializations is to restrict the search to specific regions in parameter space from symmetry arguments. This section, in particular, will restrict the numerical analysis of QAOA to 3-regular unweighted graphs (u3R) and Erdős-Rényi graphs ($G(n, q)$), which are random graphs constructed by connecting two nodes at random with probability q. Writing out the problem- and mixer-unitaries of the QAOA ansatz in its trigonometric forms, one arrives at the following terms

$$e^{i \sum_{(i,j)\in E} \frac{\gamma_l}{2} Z_i Z_j} = \prod_{(i,j)\in E} \left[\cos \frac{\gamma_l w_{i,j}}{2} + i Z_i Z_j \sin \frac{\gamma_l w_{i,j}}{2} \right] \tag{101}$$

$$e^{-i\beta_l \sum_{j\in V} X_j} = \prod_{j\in V} \left[\cos \beta_l - i X_j \sin \beta_l \right]. \tag{102}$$

Some symmetry arguments can be made from these expressions. Firstly, a shift in β by $\pi/2$ yields the same cost function. The easiest way to see this is to note that $e^{i\frac{\pi}{2}\sum_i X_i} = (X_i)^{\otimes N}$. This has the effect of flipping all the spins, which does not change the cost function due to the bit flip symmetry inherent in MaxCut. As a result, the searchable subspace for all β_l angles can be reduced to $[0, \pi/2)$.

The searchable subspace for γ_l is graph-dependent. This is evident from the notion that the problem-unitary is the product of several trigonometric terms with different

phases, dependent on $\gamma_l \omega_{i,j}$. Consider first udR graphs where the weights $\omega_{i,j} = \{0, 1\}$ and each node is connected to exactly d other nodes. For these graphs, a shift in γ_l of π results in the problem unitary being $\prod_{(i,j)\in E} iZ_iZ_j$ which is either $Z^{\otimes n}$ if d is odd or $\mathbb{1}$ if d is even. Applied to a ket-vector, these two operators will output the same ket-vector, hence creating redundancies. Therefore, one can restrict the search of all γ_l parameters to be between $[0, \pi)$ for udR graph instances. The same arguments do hold for unweighted Erdős-Rényi graph instances since the resulting operator is either Z or $\mathbb{1}$ at each qubit by similar arguments.

On the other hand, weighted graph instances are less restrictive in their symmetries. The restrictions on β_l remain; however if $\omega_{i,j}$ are different irrational numbers, a general period for the γ_l parameters cannot be found. In principle, one must search the entire real number line for potentially good solutions to MaxCut. However, one expects the phases $\gamma_l \omega_{i,j}$ to somewhat align for small γ_l, while they are effectively random for higher values. Therefore, one expects to find high-quality solutions around $\gamma_l = 0$. However, there are no guarantees that this solution is the globally optimal parameter.

Even with these restrictions, convergence to local minima has been a hurdle for many practical applications of QAOA. It, therefore, has motivated several researchers to find heuristic optimization strategies that improve the QAOA performance. In the following two subsections, two heuristic strategies will be presented. Both heuristics create efficient ways of generating initial parameters based on the optimal parameters from the previous depth $p-1$, from which all $2p$ parameters (γ_i, β_i) will be optimized using a typical classical optimizer. One of the strategies presented by Leo Zhou et al. [1] finds that the globally optimal parameters follow a linear trend from layer p to $p+1$, from which they create two heuristics. On the other hand, Lee et al. [2] conditions the new initial parameters by fixing the $2(p-1)$ parameters to be local optima of the previous depth and generates 20 random pairs of (γ_p, β_p) to find the local optima of layer p. In order to train and examine the QAOA algorithm for larger graph instances and depths more efficiently, the following two subsections present some of these authors' findings.

5.2 Interpolation Heuristic: INTERP

To address the parameter optimization problem, Leo Zhou et al. [1] introduce a heuristic approach that exploits heuristic patterns in the optimal QAOA parameters to initialize optimization efficiently. The heuristics produce quasioptima in $O[\text{poly}(p)]$ time that would require $2^{O(p)}$ randomly initialized optimization runs to surpass. Two different interpolation heuristics are constructed based on heuristic patterns in the optimal QAOA parameters. These are called INTERP and FOURIER. These heuristics generate the initial point from which to start the optimization of the $2p$ parameters at depth p.

The authors investigate the optimal QAOA parameters for MaxCut on random u3R and w3R graphs with vertex numbers $8 \leq N \leq 22$. Testing on 100 random graph instances, 10^4 random initial points were optimized to find the globally optimal parameters at depth p. The authors found that at a fixed depth p, the optimal

parameters $(\boldsymbol{\gamma_i}, \boldsymbol{\beta_i})$ had a continuous trend. In particular, for each $i = 1, 2, \ldots, p$, γ_i^* tend to increase smoothly while β_i^* tend to decrease smoothly. This pattern is similar to adiabatic quantum annealing in which the mixer-Hamiltonian is gradually turned off while the problem Hamiltonian is gradually turned on. A more detailed comparison between the two algorithms is presented in section 5.5.

From the observations, there seems to be an underlying pattern in the optimal parameters, which varies little from p to $p+1$. This pattern can be exploited where one essentially starts by finding a good set of parameters at $p = 1$ (using random initialization since the number of parameters is small). Increment to $p + 1$ and use the optimized parameter-values at the previous depth to interpolate a good initial guess for the $2(p + 1)$ parameters. The interpolation can be performed in various ways, and the authors present two methods. INTERP uses the pattern generated from depth p to linearly interpolate a guess for the parameters at depth $p + 1$. Meanwhile, FOURIER interpolates using the discrete sine/cosine transform, where the parameters at depth $p+1$ are determined from the Fourier amplitudes at depth p. This thesis limits the scope to only considering INTERP; hence only this procedure is presented. The new initial parameters at depth $p + 1$, $(\boldsymbol{\gamma}^0_{(p+1)}, \boldsymbol{\beta}^0_{(p+1)})$, are generated according to

$$\left[\boldsymbol{\gamma}^0_{(p+1)}\right]_i = \frac{i-1}{p}\left[\boldsymbol{\gamma}^L_{(p)}\right]_{i-1} + \frac{p-i+1}{p}\left[\boldsymbol{\gamma}^L_{(p)}\right]_i \tag{103}$$

where $i = 1, 2, \ldots p + 1$ and $[\boldsymbol{\gamma^L}]_i$ is the i-th component of the parametervector that corresponds to the previous depth p's optimal value, and $[\boldsymbol{\gamma}^L_{(p)}]_0 \equiv [\boldsymbol{\gamma}^L_{(p)}]_{p+1} \equiv 0$. This interpolation is correspondingly performed on β_i to find its new initial point.

It is important to stress that in both these strategies, the generated vectors $(\boldsymbol{\gamma^0}, \boldsymbol{\beta^0})$ are only educated guesses as to where the optimal parameters will lie in parameter space. The generated point is used to optimize all the $2(p+1)$ parameters at the $p+1$ level, yielding a good approximation of the optimal parameters $(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)$. Through extensive searches, Zhou et al. [1] find that their heuristic performs just as well as the best randomly initialized runs at low p, indicating that both methods find the global minima. As the depth increases, the heuristic performs better than randomized runs on average since these runs tend to converge to suboptimal solutions. The researchers found that the median number of randomly initialized runs needed to match the approximation number r of the heuristics scales exponentially with depth p.

5.3 Parameter Fixing Heuristic

As introduced by Lee et al. [2], the parameter fixing heuristic utilizes conditioned search to generate good initial points for traversing the cost landscape. Like INTERP, this method iteratively increases the circuit depth by utilizing the parameters from the previous depth. As the name indicates, when looking for an initial point at depth p, one fixes the optimized parameters at the previous layer, then generates two random points (corresponding to γ_p, β_p) then starts optimization.

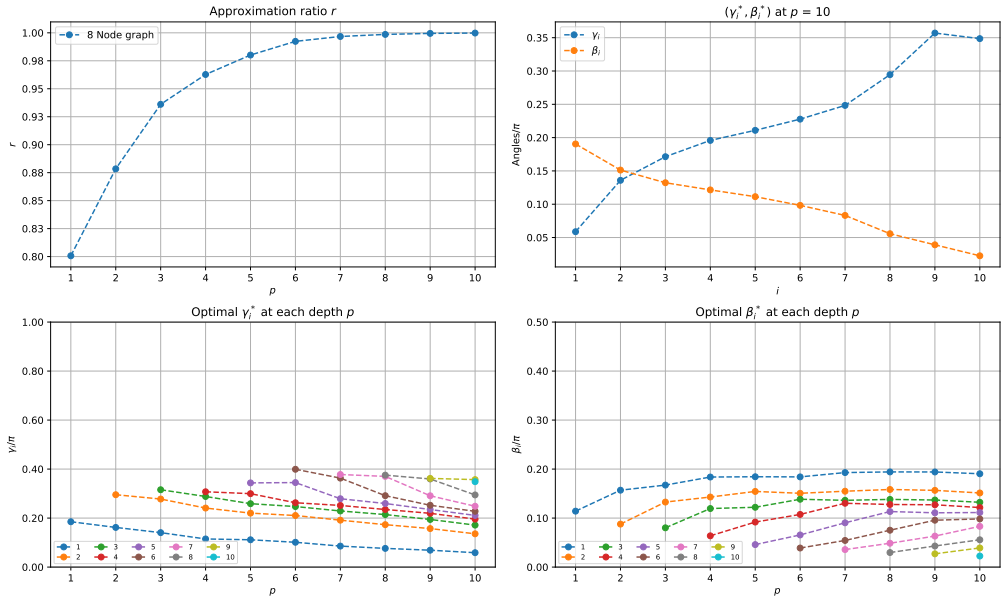
Consider here the procedure for the first three depths. The heuristic first starts at depth $p = 1$, generates two random points, and performs the optimization. This procedure is repeated 20 times in order to find the global minimum. Note that the cost landscape at the smallest depth is well-behaved, and very often, the global minima will be reached through a few random initializations alone. Alternatively, a grid search can be performed to find the optimal parameter at this depth. The cost landscape for a graph instance at the lowest layer will be provided later in this section. This procedure generates the optimal parameters (γ_1^*, β_1^*). Now the heuristic iterates to $p = 2$. When generating an initial point, (γ_1^0, β_1^0) is set to the previously found (γ_1^*, β_1^*), while the two new parameters (γ_2^0, β_2^0) are once again 2 new random points. As before, the optimizer is used to navigate the cost landscape; however, now, with four variables, after 20 new repetitions gives four new optimal parameters. These are then passed on to the $p = 3$ depth when looking for new initial points.

5.4 Results: Comparison between the heuristics

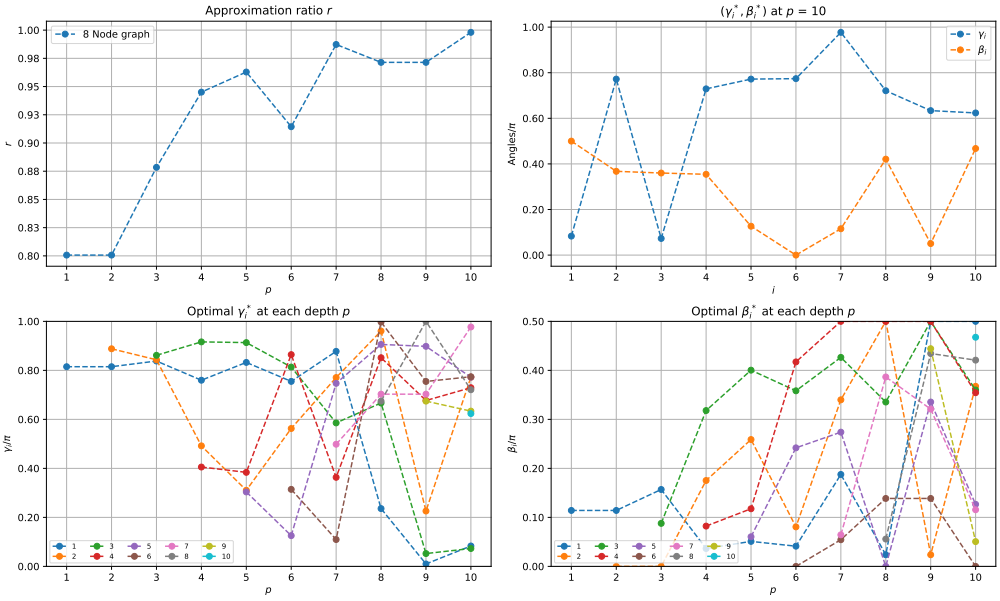
The analysis of the heuristics will be performed by combining insightful statistics from both papers. The performance measure is the approximation ratio r as a function of depth p of the QAOA circuit. This analysis method is used in both papers to evaluate the performance of their respective heuristics. However, the successes of their algorithms are explained differently. Zhou et al. [1] attribute their heuristic's performance to the closeness between the continuous pattern found in the globally optimal parameters and the parameters obtained by the heuristic. The authors do so by plotting (γ_i^*, β_i^*) with $i = 1, 2, \ldots, p$ at the final depth. On the other hand, Lee et al. [2] instead plot all the optimally found parameters (γ_i^*, β_i^*) at each depth p, essentially displaying the trajectory of each parameter in parameter space. An example of these methods is found in figure 13, where the upper left figure displays the approximation ratio, the upper right displays the optimal parameters at the final depth, while the two lower graphs display the optimal parameters at each depth p. Utilizing both methodologies, each of these heuristics will be compared to see if there are commonalities. For these simulations, the L-BFGS-B optimizer from the python library SCIPY [57] is used, an optimizer that bounds the parameters to search only within a certain area in parameter space. In these simulations, the parameter search is bounded by the symmetry arguments mentioned earlier. This optimizer is gradient-based, which can be calculated using the parameter shift method or finite differences as introduced in section 4.1. The circuits and gradients are calculated using the quantum machine learning package PennyLane [43].

5.4.1 Results from INTERP heuristic

The INTERP heuristic was tested for both u3R and Erdős-Rényi graphs. For the u3R graph type, 30 random graph instances of nodes $n = 6, 8, 10, 12, 14$ were tested, while 10 different Erdős-Rényi graph instances of nodes $n = 6, 7, 8, 9, 10$ were tested. First, ten different optimization runs were performed for each graph instance to find general patterns in the optimization. Once a pattern was established, the heuristic



(a) The statistics for a successful run.

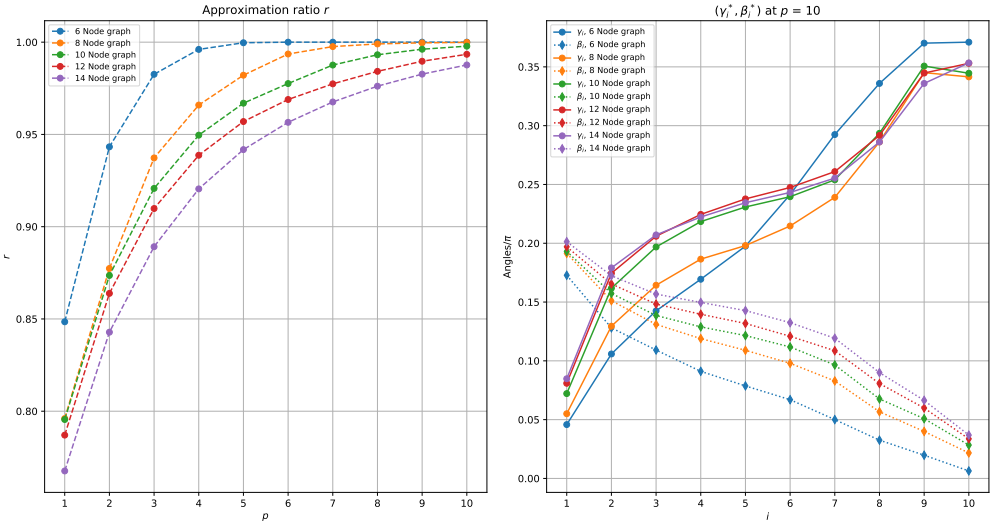


(b) The statistics for an unsuccessful run.

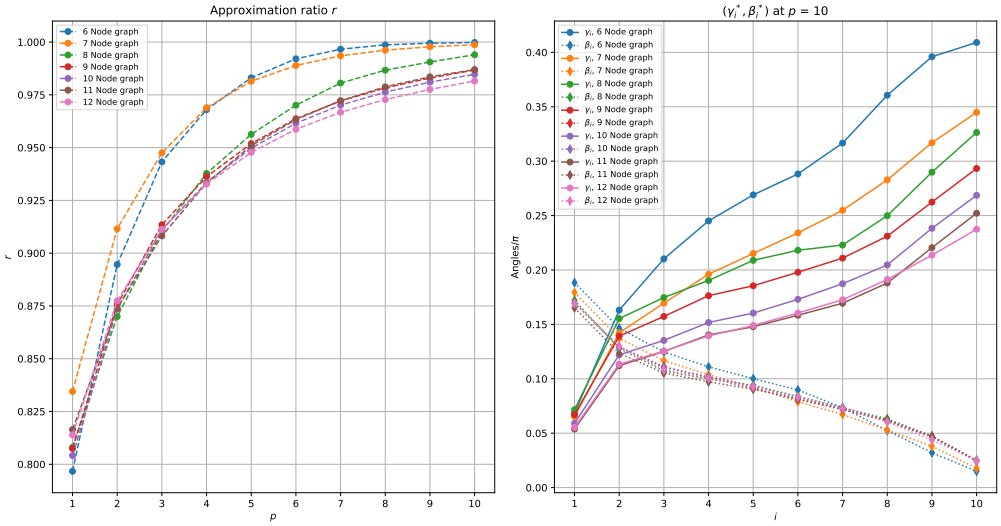
Figure 13: Two separate optimization runs on the same 8 node u3R graph instance. For each run, the approximation ratio as a function of QAOA depth p (upper left), optimal parameters (γ_i^*, β_i^*) at depth $p = 10$ (upper right) and optimal parameters (γ_i^*, β_i^*) at each depth (lower figures) are plotted.

was only run once per graph instance.

The performance of the heuristic is mixed. Consider, for instance, figure 13, where the statistics of two separate runs using INTERP are presented for the same eight-node u3R graph instance. As the figure shows, the performance is strongly correlated with the initial point used at layer $p = 1$ from which the higher depth parameters are extrapolated. For u3R graphs, the cost landscape is well behaved and has two



(a) Performance on u3R graph instances.



(b) Performance on Erdős-Rényi graph instances.

Figure 14: The average INTERP performance for both u3R (left) and Erdős-Rényi (right) graph instances. For the u3R graph instances, 30 graph instances ($n = 6, 8, 10, 12, 14$) were considered, while 10 graph instances ($n = 6, 7, 8, 9, 10, 11, 12$) were considered for the Erdős-Rényi graph instances.

global maxima and two global minima (more on the cost landscapes in section 5.4.3). The only difference between graph instances is how deep the minima and maxima are. The reason for this is that with u3R graphs, there are only a certain amount of subgraph types for $p = 1$; thus, the cost landscape is simply a weighted sum of the possible subgraphs on the particular graph instance, as explained by Fahri et al. [9].

The general trend for the well-behaved instances is that it seems to reach the global minima at the highest depth, as indicated by the upper right figure of 13a, as it follows the pattern heuristically found by Zhou et al. [1]. These simulations also have a monotonically increasing approximation ratio with increasing depth. The trend of only one particular global minima at $p = 1$ giving rise to well-behaved optimization at higher depths is seen across all graph instances. The reason why

the other global minima give rise to inconsistent results is most likely attributed to the optimizer choice. As the parameter reaches the bound of parameter space, linear interpolation will shoot the initial parameters at layer $p+1$ past the bounds, which truncates the parameter to the edge value. As a result, the parameters are forced to perform drastic changes, as evident by the lower plots of figure 13b. In this case, the approximation ratio can decrease with increasing depths, as seen in the figure. In these cases, the interpolation interpolates into a local sub-optimal minimum of a higher value than the previous layer's local minimum.

Somewhat surprisingly, this trend is not found in the Erdős-Rényi instances considered in the simulations, and most instances follow the approximation ratio-trend as indicated by 13a. With these observations in mind, when the performance of this heuristic is analyzed, the initial $p = 1$ parameter will be set to the values that consistently yield increasing approximation ratios.

Figure 14a shows the results from the simulations on random u3R graphs. As the figure indicates, the average simulation converges to an approximation ratio of 1, indicating that the optimal partitions of the graph are found. Higher depth circuits are expected to give better approximation ratios, as more subgraph types can be considered with increasing p. Additionally, as with the successful runs of figure 13a, the optimal parameters found through the heuristic at the final depth $p = 10$ follow the trend expected from Zhou et al.'s [1] paper, indicating that the found solution is close to the global one. This finding is especially promising considering that only a single run of the heuristic was performed on all 30 graph instances. Similar performance is also found for the Erdős-Rényi graph instances, as shown in figure 14b.

5.4.2 Results from the parameter-fixing heuristic

The algorithm is compared to the performance of randomly initialized runs of the same graph instances to evaluate the performance. In particular, 20 random values of the variational parameters are generated and passed onto the QAOA algorithm. The average approximation ratio is calculated, and the procedure is repeated at each depth p. As seen from figure 15, the heuristic consistently outperforms the randomly initialized runs, both for the u3R graphs and the Erdős-Rényi graph instances considered.

The graph instances considered in this paper are identical to those in Lee et al.'s [2] paper, and thus performances can be compared. The performance of the parameter-fixing technique is relatively similar between the two cases. One slight difference is that it seems that the optimizer can converge into deeper parts of the cost landscape for larger depths than for the results of Lee et al. [2], thus giving approximation ratios that are closer to 1. However, the differences are marginal. One stark contrast between these simulations and those presented by Lee et al. [2] are the results using random initialization of the QAOA parameters. Compared to their simulations, the average approximation ratio is significantly higher. For instance, in the simulation of u3R graphs at the largest depth $p = 10$, Lee et al. [2] found that only the $n = 6$ graph instance was able to surpass the $r = 0.9$ threshold in contrast to these

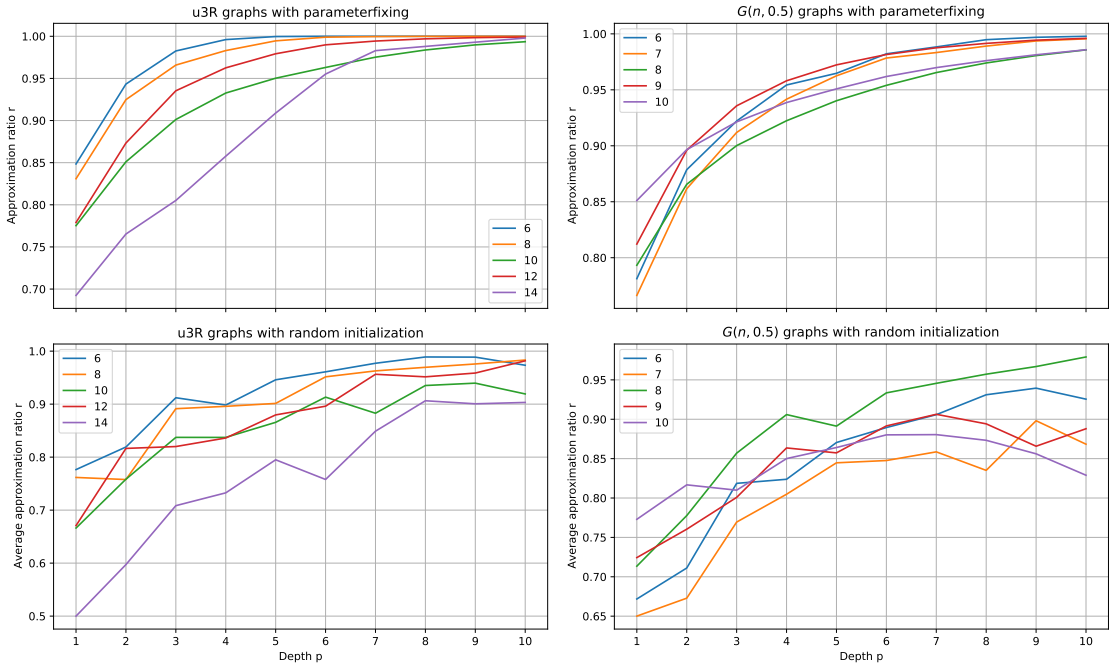


Figure 15: Comparison of the performance (measured in approximation ratio) using the two initialization methods discussed in the text. On the left are 5 graph instances of u3R graphs ($n = 6, 8, 10, 12, 14$) while the right show the performance on 5 random Erdős-Rényi ($G(n, q)$) graphs ($n = 6, 7, 8, 9, 10$) with edge probability $q = 0.5$.

simulations where all graph instances on average were able to reach it. A similar trend is observed with the Erdős-Rényi graph instances. The choice of optimizers may be the cause of this difference in performance. These simulations were run with the L-BFGS-B optimizer rather than the Nelder-Mead optimizer used by Lee et al. [2] In L-BFGS-B, gradient information is passed directly to the optimizer in order to navigate the cost landscape. This is not the case for the Nelder-Mead, a direct search method.

With the parameter fixing heuristic, one can evaluate how the optimal parameters change with increasing depths. Lee et al. [2] find that when comparing the optimal parameters of one depth to the same parameters of the subsequent depth (for instance, comparing $\gamma_2^{p=2}$ when QAOA has depth 2 to $\gamma_2^{p=3}$ at depth 3), the change in parameters is subtle. Lee et al. [2] draw their conclusions from an eight-node u3R graph. Figure 16 is an extension of their findings using the same graphs as in figure 15. Each column corresponds to the same statistics as plotted earlier in figure 13 (the approximation ratio r is not plotted here).

Somewhat surprisingly, even though the same graph instance as Lee et al. [2] studied was considered in these simulations, the trend that they found was not reproduced in these simulations, as is evident by the rapid variations of (β_i^*, γ_i^*), $i = 5, 6, 7$ for the $n = 8$ node graph. However, the trend discovered in the paper is found on the 10 and 12 node graph instances, as the two lower rows of figure 16 show. The trend is that most parameters do not change significantly when the depth is increased. This can be surprising, as the fixing of parameters only occurs when choosing the initial point for the optimization. After the point is set, all $2p$ parameters are free to

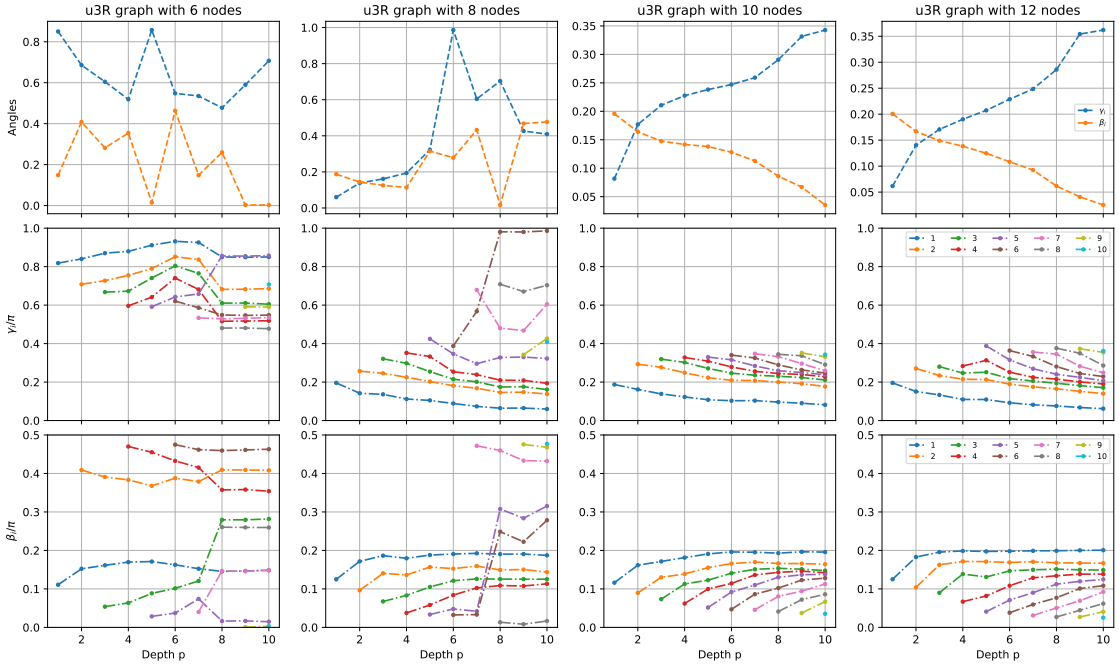


Figure 16: Comparing different statistics for the same four instances of u3R graphs ($n = 6, 8, 10, 12$) as in figure 15. The first row of plots show the optimal γ_i^*, β_i^* as a function of $i = 1, 2, \ldots, 10$ at depth $p = 10$ using the parameter-fixing heuristic. The second and third rows of graphs plots the optimal parameters γ_i^* and β_i^* respectively at each depth p using the parameter-fixing heuristic.

be optimized. After a parameter has relaxed to its fixed value, further optimization changes it little. This effect is especially prominent at higher depths, particularly $p \geq 6$, where parameters that change most significantly are the newly introduced ones.

This fixing of the parameters can be intuitively understood by noting that the optimization at depth p is a constrained version of the optimization at $p+1$. Considering this fact, in tandem with the observation that the approximation ratio changes little at higher depths (as it is already close to $r = 1$), setting the newly introduced parameters to zero gives the same result as the previous depth while finding the minimum of the constrained cost landscape of the newly introduced parameters yields lower costs. In these simulations, the trajectories of the parameters are less fixed than the findings from Lee et al. [2].

An interesting find is that for the well-behaved instances, the optimal parameters at depth $p = 10$ follow the trend found in INTERP for the optimal parameters, namely that γ_i^*, β_i^* at the highest depth p follow the continuous trend of increasing γ_i and decreasing β_i. The finding suggests that out of the 20 new pairs of points at layer $p + 1$, the pairs that yield better approximation ratios follow a similar trajectory as the parameters of the INTERP heuristic. Comparing the two lower rows of figure 16 with the lower row of figure 13a, it seems that the fixing of the previous layer's parameters is prevalent in the INTERP heuristic as well, especially at higher depths p.

However, the parameter-fixing heuristic searches larger parts of parameter space due to the 20 randomized parameters at each depth p. As a result, the method is less prone to getting stuck in local minima than the INTERP method, which can interpolate into local minima and get stuck, as seen in figure 13b. This problem is circumvented in the parameter-fixing heuristic as indicated by figure 15 where the monotonicity in the increasing approximation ratio remains even if the optimal parameters (γ_i^*, β_i^*) do not necessarily follow the trend predicted by Zhou et al.[1].

Naively using the INTERP heuristic might give mixed results, as indicated earlier, in contrast to the parameter-fixing heuristic, which generally performs well for all parameter initializations. However, the parameter fixing heuristic is slower since its runtime is of $\mathcal{O}(np)$ where n is the number of sampled pairs (γ_p, β_p) at each depth p (this was set to 20 in these simulations). This scaling is in contrast to the INTERP heuristic, where the interpolated initial parameters are optimized only once at each depth p, which scales like $\mathcal{O}(p)$. However, suppose the correct optimal minima at $p = 1$ are chosen. In that case, the INTERP heuristic seems to give a monotonically increasing approximation ratio with increasing depths and generally converges to a good optimal solution. As a result, the INTERP heuristic generally seems to outperform the parameter fixing heuristic.

5.4.3 Cost Landscape

With the observation that fixed parameters tend to remain fixed at higher depths, it is possible to get a glimpse into how the cost landscape changes as the depth is increased. Figure 17 presents the cost landscape for a ten node u3R graph for QAOA depths of 1 and 2. The right subfigures of figure 17 present the cost landscapes as a function of the (γ_2, β_2) parameters for the particular cases where (γ_1, β_1) are fixed to the minimum and maximum of the previous depth as indicated with diamonds in figure 17. The orange diamond is the $p = 1$ minimum that consistently gives rise to well-behaving runs for the INTERP heuristic, while the green diamond is one of two potential maxima. Note here that the cost-landscape corresponds to expectation values of the negative cost Hamiltonian, which is needed to use the SCIPY minimize library as it optimizes in the negative cost direction. Thus, blue regions correspond to better partitions of the graph instance. As the figure shows, fixing the previous layer parameters substantially affects the cost landscape. There are two things to take note of with these cost-landscapes in particular.

Firstly, as the previous layer parameters are fixed to the optimal values of layer $p = 1$ (lower right of figure 17), large regions of the cost landscape yield reasonable costs, as the areas of the cost landscape are predominantly blue. Additionally, the attractive basin around the global cost is large; thus, most initialization within this area will converge to the constrained global minimum. Note that if the optimization was performed without bounds, the constrained global minimum for this particular depth would be reached as the minima are degenerate in parameter space. If the searchable parameter space is restricted, as in these simulations, there are two low-cost minima to converge towards.

Secondly, suppose a maximum of the previous layer is fixed instead. In that case, a

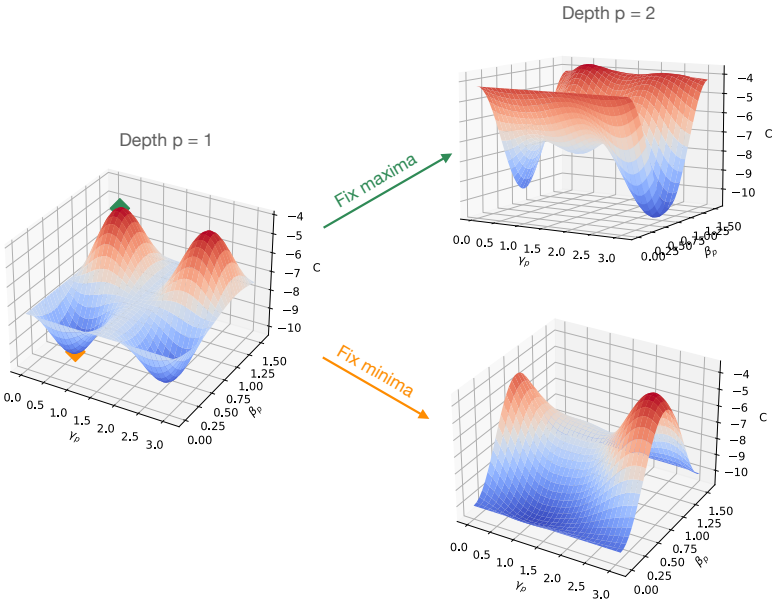
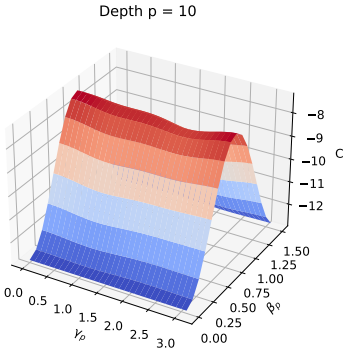


Figure 17: Surface-plots of the cost landscape, $-\langle\psi(\boldsymbol{\gamma},\boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle$, for layers $p = 1$ (left) and $p = 2$ (right). For the $p = 2$ upper (lower) figure, the previous parameters $(\gamma_1, \beta_1\ )$ are fixed to the arguments of the orange (green) diamonds of $p = 1$. Blue regions corresponds to better expected cuts of the graph. These particular cost landscapes was generated using a u3R 10-node graph.

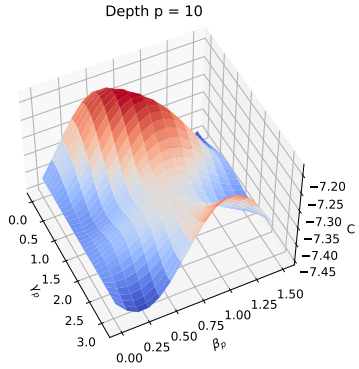
third suboptimal minimum emerges, hinting at the notion of a non-convex landscape as the previous layer parameters are changed. It should also be noted that two of the three minima are suboptimal solutions if the parameter space is bounded. Hence the attractive basin of the constrained global minimum is relatively small compared to when the minimum is fixed. Naturally, this discussion is incomplete since there are more degrees of freedom to optimize, which are not considered as they are fixed in these figures. From the previous figures of the parameter trajectories, it is recognized that usually, all parameters are optimized at the lower depths.

In figure 18 the expected cost $-\langle\psi(\boldsymbol{\gamma},\boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle$ for a 10-layered circuit is plotted as a function of the final depth parameters $(\gamma_{10}, \beta_{10})$. The rest of the $2(p-1)$ parameters remain fixed, where figure 18a sets the fixed values to the values found from repeated application of the parameter-fixing heuristic. In figure 18b, the fixed parameters are random values sampled uniformly within the bounds of the QAOA parameters. The sampling procedure was performed 25 times and the cost averaged. In these cases, a ten node graph was considered with an optimal MaxCut value of 13.

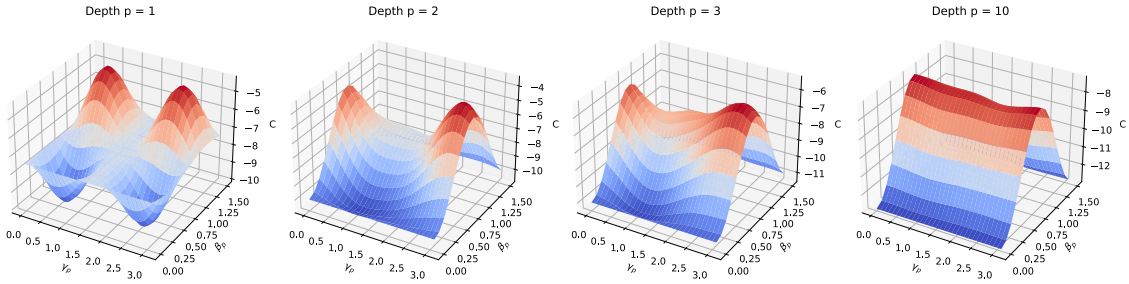
Working with the assumption that the parameters of the previous depths remain fixed, the reason for the excellent performance becomes evident. Lee et al. [2] attribute the positive performance of the parameter-fixing heuristic to the appearance of these minima-lines, which becomes deeper with increasing depth, as shown in figure 18c. As the figure shows, the transition from the simple local minimum at depth $p = 1$ to the lines of minima happens reasonably early. However, these lines are not deep enough to give high approximation ratios until intermediate depths ($p \geq 6$).



(a) The cost-landscape with all previous parameters fixed to the optimal value.



(b) The average cost landscape where the fixed parameters are set to random values.



(c) The cost landscape plotted as a function of the last parameters (γ_p, β_p) plotted for various depths. The $2(p-1)$ fixed parameters follow from the parameter-fixing heuristic.

Figure 18: The cost $-\langle\psi(\boldsymbol{\gamma},\boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle$ plotted as a function of the last parameters $(\gamma_{10}, \beta_{10})$ with fixed optimal parameters (left) and random fixed parameters (right). Further explanation of the figure is found in the text.

As is evident by these cost landscapes, the cost function essentially becomes convex within the bounds set for the parameters.

If a similar procedure were to be performed using randomly fixed parameters, the cost would concentrate around local optima, far off the global one, as shown in figure 18b. However, it needs to be emphasized that this argument does not hold as there are several degrees of freedom to optimize. The performance of the randomly initialized parameters of figure 16 indicates that good solutions indeed are found.

5.5 Difference between QAOA and Quantum Annealing

In order to understand QAOA and its mechanisms, it is informative to compare it to adiabatic quantum computation, which usually goes by the name Quantum Annealing (QA). There are slight differences between these two terms; however, they usually lie in the fact that Quantum annealing cannot sufficiently approximate the adiabaticity required of adiabatic quantum computing.

The goal of adiabatic quantum computation is to find the ground state of a problem

Hamiltonian H_P, which it does by adiabatically evolving from the ground state of a known Hamiltonian H_B at time $t = 0$ to the problem Hamiltonian H_P at time $t = T$ using the following time-dependent Hamiltonian

$$H_{\mathrm{QA}}(s(t)) = sH_P + (1-s)H_B, \quad s(0) = 0,\ s(T) = 1 \tag{104}$$

This method is grounded in the adiabatic theorem, which states that *a physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum* (Born, 1928 [58]). Therefore, at $t = T$, the system is in the ground state of the problem Hamiltonian, and thus the problem's solution can be found. The time-evolution of a quantum state can in general be written as a unitary $|\psi(t)\rangle = U(t)|\psi(0)\rangle$. Inserting this into the time-dependent Schrodinger equation gives the following differential equation for the unitary operator:

$$i\frac{\partial U(t)}{\partial t} = H(t)U(t) \tag{105}$$

$$U(t) = e^{-i\int_0^t H(t)dt} \tag{106}$$

In the second equation it was implicitly assumed that $[H(t_1), H(t_2)] = 0$. In the case that the Hamiltonian is time-independent, one would get $U(t) = e^{-iHt}$. Using the time-evolving operator, one can decompose the evolution from time $t = 0$ to $t = T$ as follows:

$$U(T,0) = U(T, T-\Delta t)U(T-\Delta t, T-2\Delta t)\dots U(\Delta t, 0) \tag{107}$$

In general, the time-intervals Δt_i can be different. Choosing Δt to be sufficiently small such that $H(t)$ is constant in each of these intervals yields an approximate expression for the time evolution operator,

$$U(j\Delta t, (j-1)\Delta t) = e^{-i\int_{(j-1)\Delta t}^{(j)\Delta t} H(t)dt} \approx e^{-iH(j\Delta t)\Delta t}, \tag{108}$$

where $H(j\Delta t)$ is the constant Hamiltonian in the time-interval between times $(j-1)\Delta t$ and $j\Delta t$. Decomposing the total time evolution into p parts yields

$$U(T,0) = \prod_{j=1}^{p} U(j\Delta t, (j-1)\Delta t) \approx \prod_{j=1}^{p} e^{-iH(j\Delta t)\Delta t} \tag{109}$$

Now the evolution operator is expressed as a product of time-independent evolution operators over p small time intervals. However, since H is given by H_{QA}, the terms

in the Hamiltonian do not commute ($[H_P, H_B] \neq 0$ by design). However, one can employ the Trotter-Suzuki decomposition to address this issue, which to first-order states that $e^{it(A+B)} = e^{itA}e^{itB} + O(t^2)$ [59]. Using this decomposition and inserting for the hamiltonian H_{QA} yields

$$U(T,0) = \prod_{j=1}^{p} e^{-i(1-s(j\Delta t))\Delta t H_B} e^{-is(j\Delta t)\Delta t H_p} \tag{110}$$

By setting the QAOA parameters to be $\gamma_j = s(j\Delta t)\Delta t$ and $\beta_j = (1 - s(j\Delta t))\Delta t$, one could simulate quantum annealing using the QAOA ansatz. As a result, one can view QAOA as a generalization of quantum annealing. QAOA differs significantly from QA in one aspect, namely adiabaticity. In QA, one starts in the ground of some known Hamiltonian and adiabatically evolves this state in such a way that it always remains in the instantaneous ground state of the time-dependent Hamiltonian $H_{\mathrm{QA}}(t)$. This is represented by the $s(t)$ function (usually given as a linear interpolation $\frac{t}{T}$), which gradually introduces the problem-Hamiltonian H_P during the evolution. In contrast, QAOA does not gradually introduce the problem Hamiltonian but instead switches on the mixer and problem Hamiltonian for varying durations β_i and γ_i, respectively. As a result, QAOA is not restricted to following an adiabatic path in these parameters. Zhou et al. [1] compared QAOA with QA and found that the QAOA procedure can find optimized annealing paths and exploit diabatic transformations to increase the ground state population through classical optimization. This allows QAOA to find good approximate solutions faster than QA, with an algorithmic run time that scales like $T = O(1/\Delta_{min}^2)$. Δ_{min} is the minimum spectral gap, i.e., the minimal gap between the ground state and the first excited state through the annealing process.

To summarize, QAOA is a particular algorithm within the family of VQAs which use alternating unitaries to evolve the state towards the minimum cost. In this section, the algorithm was applied to the combinatorial optimization problem MaxCut on u3R and Erdős-Rényi graph instances, where it is found that the algorithm is indeed effective in finding the ground state spin configuration that minimizes the cost. Similarities with quantum annealing are further discussed in order to get an understanding of the mechanisms of the algorithm. Here it is found that the algorithm can find diabatic paths in parameter space that cause faster convergence to the ground state energy through classical optimization. As indicated by the optimization from randomized initial points in parameter space, the cost landscape is non-convex, and the optimizer often gets stuck in local minima. Two initialization heuristics were introduced to combat this, successfully converging towards the global minimum costs. Note that training these algorithms take a considerable amount of time, within which multiple runs of the classical Goemans-Williamson algorithm can be run to achieve minimum costs with a high probability.

Through the lens of QAOA, this section has also exemplified how the general structure of variational algorithms is applied in a general hybrid quantum-classical computational framework. In particular, a classical optimizer can train the variational parameters in quantum circuits using a black-box approach where inquiries are made

to the quantum computer to estimate gradient directions. From this point forward, two different approaches will be explored further. The first utilizes a classical feedforward neural network to learn the optimal QAOA parameters based on the INTERP heuristic. The other approach attaches a feedforward neural network at the end of quantum measurements to escape local minima in the QAOA cost landscape.

6 Using neural networks to find the optimal parameters

Closing this part of the thesis, the findings presented so far will be used to create a machine learning procedure that consistently produces good mean cost values for the MaxCut problem. Section 5 established that the optimal QAOA parameters (γ_i, β_i) at a given p follow increasing and decreasing trends, respectively. Using this knowledge, Alam et al. [3] propose using classical machine learning techniques to learn the optimal parameters at a given depth. Using this pre-trained machine learning algorithm, one may use it to predict the optimal parameters on unseen graph instances using optimal parameters at low depth. Their approach involved using Gaussian processes to predict the optimal parameter values. In anticipation that neural networks will be used later in the thesis, this section extends their approach by using a neural network to predict the $2p_f$ optimal parameter values at the final depth using only the optimal parameters at level $p = 1$. Figure 19 illustrates how this procedure works.

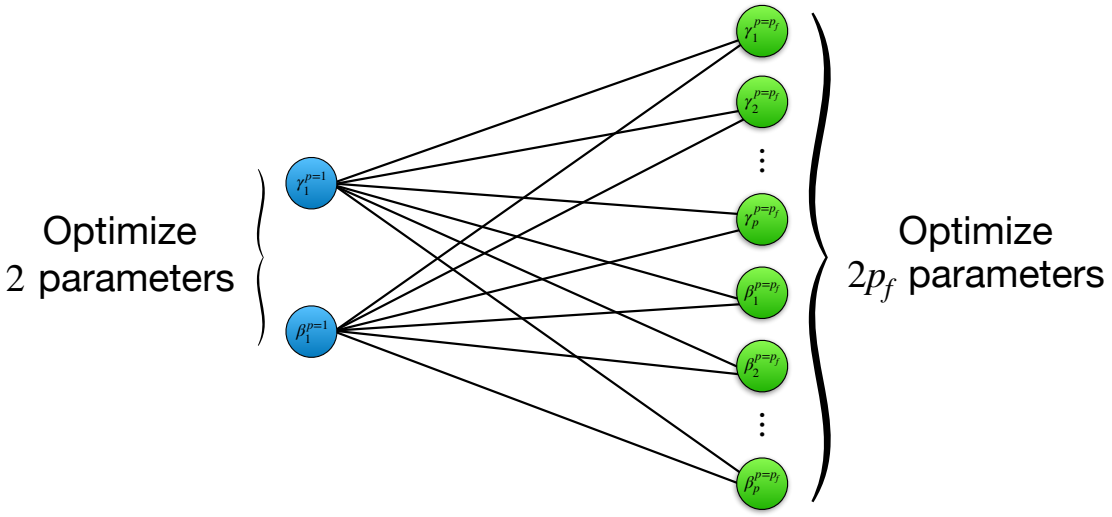


Figure 19: A figure explaining how one incorporates machine learning in this procedure. Using the optimized QAOA parameters at $p = 1$, $(\boldsymbol{\gamma_1}, \boldsymbol{\beta_1})$, the neural network predicts all the QAOA parameters at the final depth $p = p_f$. Hence the model takes two parameters as input and outputs $2p_f$ parameters. Using the predicted values as an initial guess, a final optimization is performed to find the optimal parameters $(\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*)$ at depth $p = p_f$.

This approach consists of two parts. The first part is generating a set of data to learn the optimal patterns. The dataset consists of all optimal parameters at QAOA depths ranging from 1 to 8 for 200 different graph instances. These optimal parameters were generated using an identical setup as presented in section 5 using the INTERP heuristic. The graph dataset consisted of various twelve node w3R graph instances with weights sampled uniformly between $[0, 1]$. From here, the dataset is separated into a 20-80 train-test split to show that a significant amount of graph instances is not needed to form good results. The dataset is then used to train a neural network. The neural network that is considered here is rather simple:

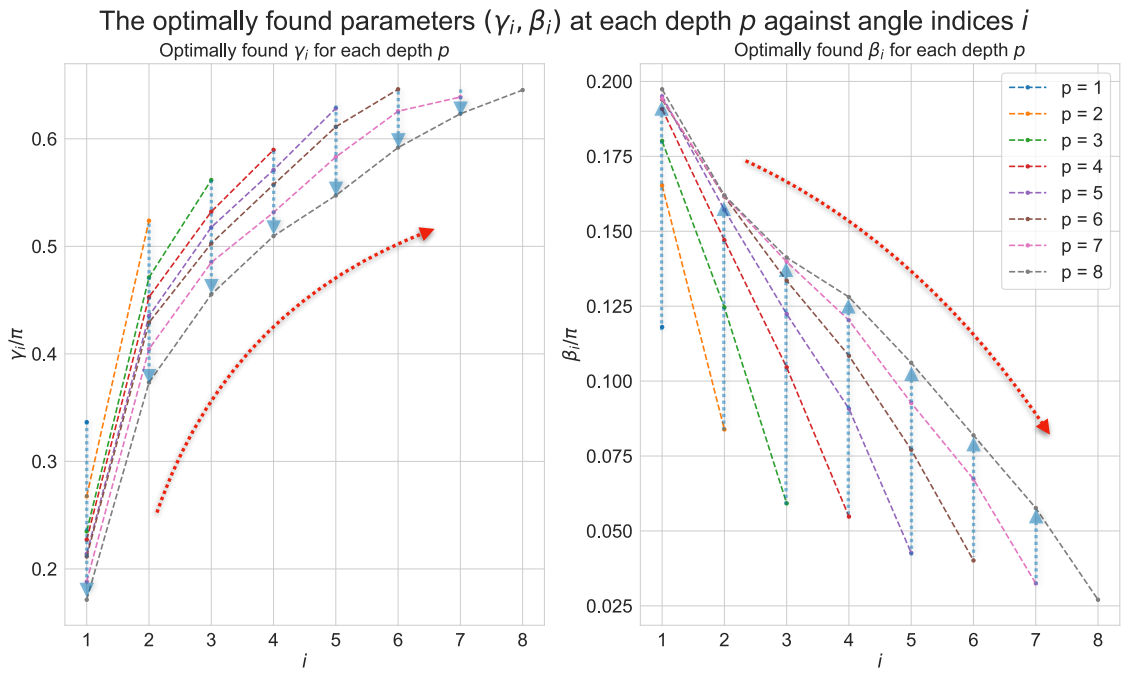


Figure 20: This is an example showing the trends as indicated in the paper. Notably, for each angle index i, as the depth of the QAOA circuit increases, the optimal angle γ_i decreases while β_i increases, as highlighted by the blue arrows at each angle index i. Also, for a fixed p, γ_i tends to increase while β_i tends to decrease with increasing angle index i as highlighted by the red arrows.

it consists of a single input layer and a single output layer. The input layer has two nodes, while the output layer has $2p$ nodes. The neural network takes as input the optimal (γ, β) at depth $p = 1$, then outputs a set of parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ that can be used as initialization for a regular optimization run.

There are two particular trends in the parameters that are worth taking a note of. The first is the familiar trend of increasing γ_i and decreasing β_i at a given depth p. The second is that given an angle index i, the parameter γ_i decreases when increasing the circuit depth p while β_i increases with increasing p. Figure 20 highlights this trend using arrows where the red arrows represent the former trend while the blue arrows represent the latter. This is the motivation for trying to use the optimal parameters at depth $p = 1$ to find the optimal $2p$ parameters at higher depths. For instance, if one were to predict the initial parameters at $p = 2$ using the $p = 1$ optimal parameters $(\gamma_1^{p=1}, \beta_1^{p=1})$, the intuitive values would be to set $\gamma_1^{p=2}$ to be lower than $\gamma_1^{p=1}$ and set $\gamma_2^{p=2}$ to be larger than $\gamma_1^{p=2}$ based on the observed trends shown in 20. One would correspondingly do the same for $\boldsymbol{\beta}^{p=2}$, and continue in a similar fashion to predict the optimal parameters at higher depths.

In order to quantify the expected predictive capability of the model, the correlation between the model's features and expected output was calculated using data from the generated dataset. The results are shown in figure 21. In this figure, the output is shown along the x-axis, while the correlation between this output at the various features of the model is plotted along the y-axis. p is included as a feature of the model since the size of the neural network is determined by the QAOA depth p.

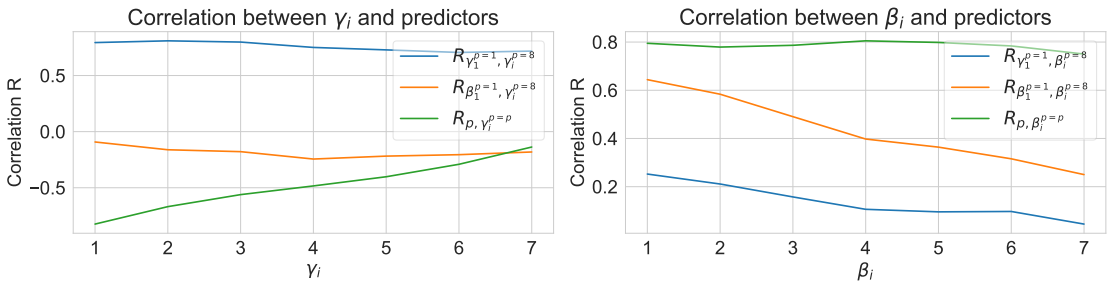


Figure 21: Correlations between the predictors and the response variables found from the generated dataset. The outcomes $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are plotted along the x-axis and the correlation between it and the predictor variables along the y-axis. Alternatively, outcome b in $R_{a,b}$ from the legend in the plots are plotted along the x axis, and the correlation between it and the feature a is plotted along the y axis.

Firstly, as expected, the correlation between γ_i (β_i) and p is negative (positive) as per the earlier findings indicated using the blue arrows in figure 20. The predictive power of the neural network features seems to vary between the types of variables. As the figure 21 shows, $\gamma_1^{p=1}$ correlates highly with the all the gamma-angles to be predicted at $p = 8$, hence this feature can be efficiently used to predict all the gamma angles $\boldsymbol{\gamma}^{p=8}$. This cannot be said for the $\boldsymbol{\beta}^{p=8}$ where it is found that the features used as input to the neural network lose predictive power with increasing angle index i. They still correlate with the outputs, however, to less degree than for the lower indices.

As figure 22 shows, the model is able to extrapolate the optimal parameters from the 40 graph instances used to train the model. Since a heavy linear trend is associated with the parameters (increasing γ_i, decreasing β_i) and a linear trend between the outputted parameters and input features, fitting a linear model to the data gives good results as expected. This is particularly the case for the predicted γ_i parameters, as the neural network can capture the width of the parameters. In the anticipation that the procedure places the initial parameter vector close to the global minimum of the optimization landscape, the predicted parameter values are used as the starting point for an optimization run. This procedure was tested for final depths $p_f = 5, 6, 7, 8$ and the results are presented in table 2.

p	Mean r	Std. deviation
5	0.944	0.014
6	0.957	0.012
7	0.967	0.011
8	0.974	0.011

Table 2: A table showing the mean approximation ratio r and the associated standard deviation when the above method is used as the initialization procedure for depths $p = 5, 6, 7, 8$ for the various graphs in the test dataset.

As figure 23 shows, the prediction in the optimal QAOA parameters can have large relative errors associated with them, particularly in the case of the β_i predictions.

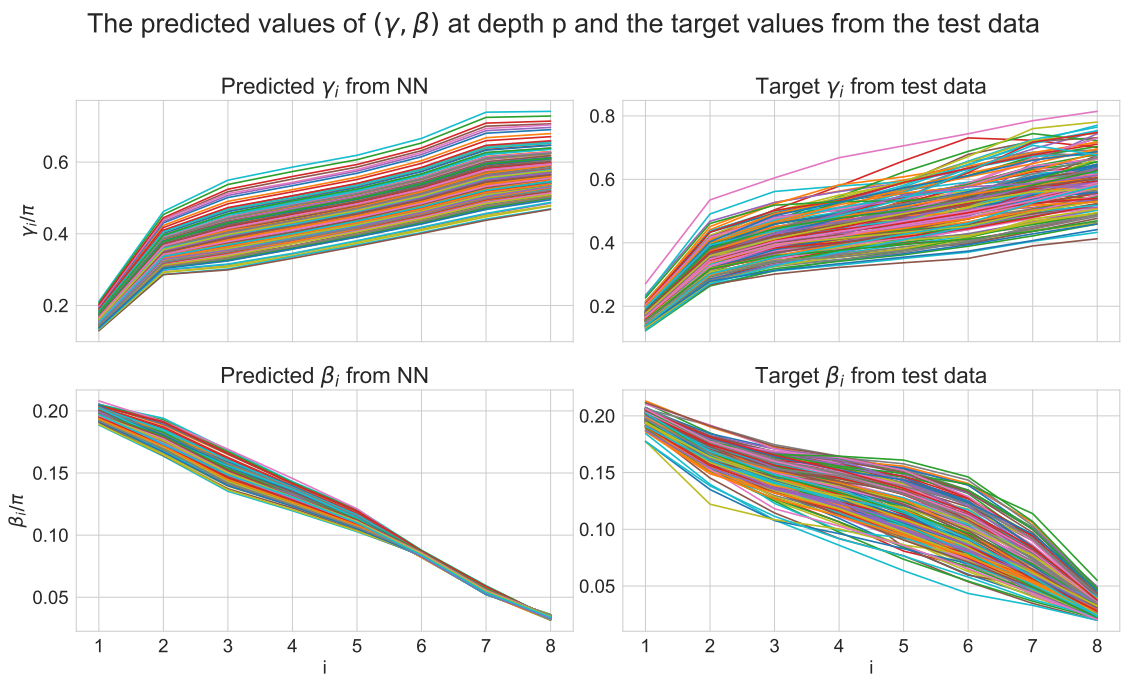


Figure 22: A plot showing the predicted output of the linear model when applied to 160 separate graph instances. The input to the model is the optimal parameters (γ, β) at depth $p = 1$ while the output are the 16 parameters associated with the procedure at $p = 8$. The first column shows the predicted initialized values of (γ_i, β_i) from the linear model, while the second column shows the optimal (γ_i, β_i) when INTERP is used on the same graph instances. The model was only trained on 40 of the 200 graph instances that constituted the entire graph dataset.

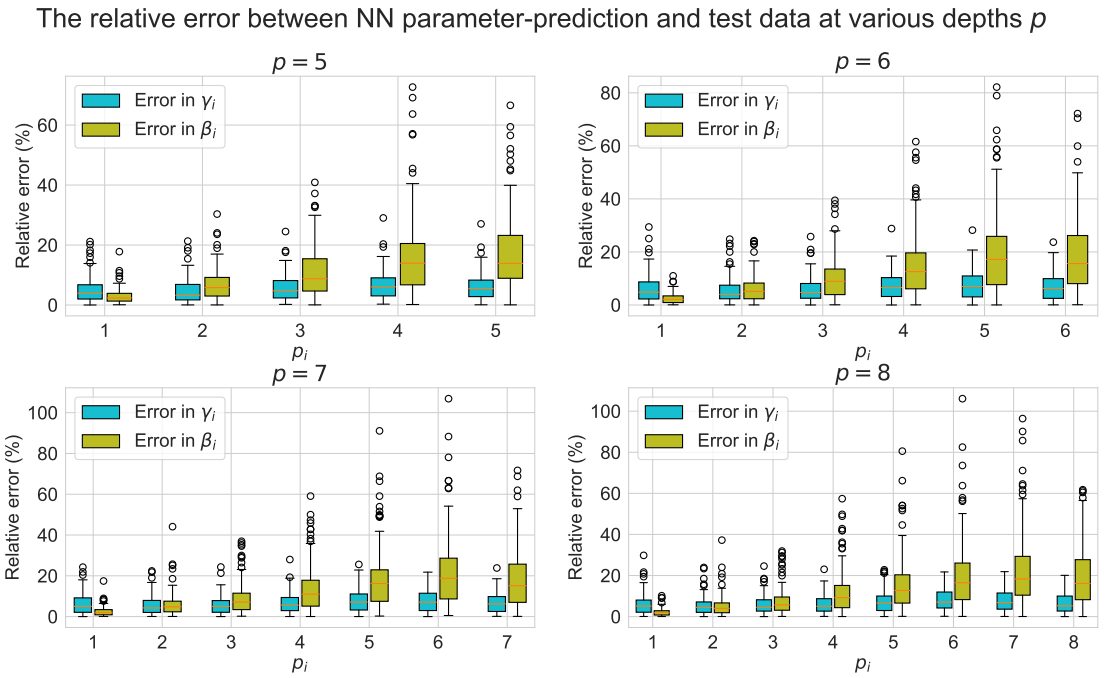


Figure 23: The relative error between the predicted parameter and the global minimum found in the test dataset for the various QAOA depths $p = 5, 6, 7, 8$.

This is to be expected as the correlations show that there is only a weak linear trend between the $p = 1$ and $p = 8$ parameters as the angle index i at $p = 8$ increases. Therefore, as expected from the trend in correlations, the relative error increases with increasing angle index i for the β_i parameters and remains relatively constant with γ_i. The effect of these relatively large errors in β_i predictions can also be seen in figure 22. The ML predictions can capture the wide span in the optimal parameters in γ_i for the various graphs. However, the model is incapable of doing so with the β_i parameters, causing relatively large errors. However, despite the predictions in β_i having inaccuracies, the initial point is still within the attractive basin of the global optimum. Therefore, if one were to use the predicted parameters as the initial point for optimization, the optimization would easily converge to the same global minimum found in the test dataset, yielding performances as noted in table 2.

As this section has shown, the procedure of using machine learning to learn the optimal parameters of QAOA seems effective. This further validates the INTERP heuristic's effectiveness. Through this method, one only needs to perform the $p = 1$ optimization to interpolate all the $2p_f$ parameters at the final depth from which a final optimization run is performed to correct the prediction. However, there are limitations to this method, as shown by the relatively large errors in the β_i predictions stemming from its low correlations with the input features of the neural network. One potential workaround to interpolate to further depths could be incrementally increasing the number of features in the neural network. For instance, if one were to interpolate the parameters from $p = 1$ to $p = 15$, one could first interpolate to $p = 8$ using the above procedure. Then, using the 16 found parameters as a set of features, one could train a neural network with 16 input nodes and 30 output nodes to perform the final prediction. Alternatively, one could look for other potential features with stronger correlations with the final parameters.

The following section presents the final approach that the thesis considers: using a neural network to escape from a local minimum once encountered. Similarly to this procedure, a feedforward neural network is considered; however, it is attached to the measurements at the end of the quantum computation instead.

7 Avoiding local minima in VQA with neural networks

The previous sections have either exclusively considered machine learning or simple VQA procedures where the outputs of the quantum systems are classically optimized. Most of the latter problems keep the cost landscape fixed and use momentum, adaptive step sizes, and randomization to escape local minima. This section seeks to utilize the machinery from machine learning actively on VQA procedures to escape local minima. An overview of this procedure is shown in figure 24. For this purpose, two heuristics are developed by Riveradean et al. [4] where the cost-landscape itself is modified in order to avoid local minima.

The modification is performed by attaching a classical single-layer feedforward neural network at the end of the quantum circuit measurement. The introduction of a neural network adds additional parameters to the model and thus increases the number of parameters. As discussed in more detail in section 4.3, increasing the number of parameters has been linked to improved trainability of variational circuits as expected from the cost landscapes of classical neural networks [13]. The cost landscape can be continuously deformed as a function of its weights by post-processing the measurement output through the neural network. Depending on how this cost landscape is deformed, Riveradean et al. [4] create two different algorithms. In the ESCAPE algorithm, the deformation is applied once the optimization of the VQA parameters encounters a local minimum. The second algorithm, GUIDE, iteratively optimizes the cost landscape through the entire optimization as a means of continuously evading local minima. The thesis considers only the ESCAPE algorithm in more detail.

The paper considers a general variational problem,

$$\min_{\boldsymbol{\theta}} \langle \psi(\boldsymbol{\theta}) | H_D | \psi(\boldsymbol{\theta}) \rangle \tag{111}$$

$$H_D = \sum_{\boldsymbol{x} \in \{0,1\}^N} C(\boldsymbol{x}) |\boldsymbol{x}\rangle\langle\boldsymbol{x}| \tag{112}$$

where H_D is a diagonal Hamiltonian in the computational basis and $C(\boldsymbol{x})$ is some cost function that maps the input $\boldsymbol{x} \in \{0,1\}^N$ to some output in $\mathbb{R}$. This is usually how a classical optimization problem is presented for VQA problems; an example was presented when applying QAOA on MaxCut in section 5. However, the VQA architecture needs to be combined with feedforward neural networks for this hybrid classical-quantum model, as shown in figure 24.

The architecture of the classical-quantum hybrid model is as follows: Prepare the N qubit quantum parametrized state $|\psi(\boldsymbol{\theta})\rangle$ according to an ansatz of choice and perform a computational basis measurement of the circuit, resulting in a classical bitstring $\boldsymbol{x} \in \{0,1\}^N$. This bitstring can be turned into a string of $\{-1,1\}^N$ by the transformation $\boldsymbol{x}' = 2\boldsymbol{x} - 1$. This string is the input to a single layer, fully connected feedforward neural network with weights $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ (no bias term is

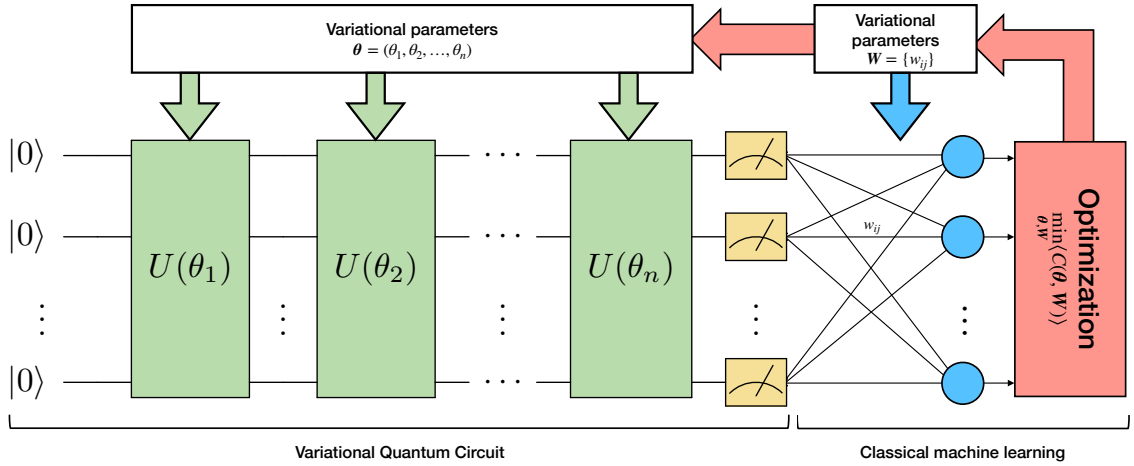


Figure 24: A schematic overview of how the neural network is included in this procedure.

included), hence maps the string $\{-1, 1\}^N$ to $\mathbb{R}^N$. The resulting vector is now fed through an activation function, which in this case is the tanh function, resulting in the output vector $f_W(\boldsymbol{x}) = \tanh(\boldsymbol{W}(2\boldsymbol{x} - 1)) \in \mathbb{R}^N$. The activation function is a relaxed sign function; however, the cost function in the case of MaxCut accepts strings of $\{-1, 1\}^N$ as input, as noted in equation 94. Therefore, a final $\text{sgn}(x)$ function is applied to all the elements in the vector, resulting in a vector of $\{-1, 1\}^N$. Therefore, the whole procedure maps a binary basis measurement string $\{0, 1\}^N$ into a string of $\{-1, 1\}^N$, on which the cost function C is evaluated. There are two ways of viewing this combination of classical and quantum processing. One way is to view the quantum circuit as a sample generator that produces a bit string $\boldsymbol{x}$ with probability $p(\boldsymbol{x}|\boldsymbol{\theta})$ dependent on the final state prior to measurement. In this case, the mean cost can be expressed as

$$\mathcal{C}(\boldsymbol{\theta}, \boldsymbol{W}) = \sum_{\boldsymbol{x} \in \{0,1\}^N} p(\boldsymbol{x}|\boldsymbol{\theta}) C\left(f_W(\boldsymbol{x})\right) \tag{113}$$

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = |\langle \boldsymbol{x}|\psi(\boldsymbol{\theta})\rangle|^2. \tag{114}$$

This expression can be rewritten in order to get a different interpretation more aligned with traditional VQA problem descriptions

$$\mathcal{C}(\boldsymbol{\theta}, \boldsymbol{W}) = \sum_{\boldsymbol{x} \in \{0,1\}^N} \langle \boldsymbol{x}|\psi(\boldsymbol{\theta})\rangle\langle\psi(\boldsymbol{\theta})|\boldsymbol{x}\rangle C\left(f_W(\boldsymbol{x})\right) \tag{115}$$

$$= \sum_{\boldsymbol{x} \in \{0,1\}^N} \langle\psi(\boldsymbol{\theta})|\left(C\left(f_W(\boldsymbol{x})\right)|\boldsymbol{x}\rangle\langle\boldsymbol{x}|\right)|\psi(\boldsymbol{\theta})\rangle \tag{116}$$

$$= \langle\psi(\boldsymbol{\theta})|H(\boldsymbol{W})|\psi(\boldsymbol{\theta})\rangle, \tag{117}$$

$$H(\boldsymbol{W}) = \sum_{\boldsymbol{x} \in \{0,1\}^N} C\left(f_W(\boldsymbol{x})\right)|\boldsymbol{x}\rangle\langle\boldsymbol{x}| \tag{118}$$

From this expression, it is evident that the parameters of the classical neural network impact the variational parameters' cost landscape due to the $\boldsymbol{W}$-dependence of the Hamiltonian. Since the model involves both quantum and classical parameters, the differentiation of the cost with respect to the quantum variables, $\nabla_{\boldsymbol{\theta}} C\left(f_W(\boldsymbol{x})\right)$, and the classical variables, $\nabla_{\boldsymbol{W}} C\left(f_W(\boldsymbol{x})\right)$, need to be evaluated. Using (117) as the cost function enables the use of the parameter-shift rule to evaluate the gradient as presented in section 4.1. Using equation (113), the derivative of the cost with respect to the classical parameters can be expressed as the sum of gradients over all the samples from the variational circuit

$$\nabla_W \mathcal{C}(\boldsymbol{\theta}, \boldsymbol{W}) = \sum_{\boldsymbol{x} \in \{0,1\}^N} p(\boldsymbol{x}|\boldsymbol{\theta}) \nabla_W C\left(f_W(\boldsymbol{x})\right). \tag{119}$$

Using $\boldsymbol{x}$ as samples, the gradient $\nabla_W C\left(f_W(\boldsymbol{x})\right)$ is calculated through backpropagation as presented in section 2.5.2. In particular, one fixes the circuit parameters $\boldsymbol{\theta}$ and sample bitstrings from the circuit, which are used in the error terms as they are backpropagated through the neural network. The quantum resources needed to calculate this gradient are insignificant compared to calculating the gradients of gate parameters using parameter-fixing. As explained in section 4.1, the scaling of calculating $\nabla_{\boldsymbol{\theta}}$ using the parameter-shift rule is $2PS(R+1)$. On the other hand, the number of samples from the quantum computer needed to estimate $\nabla_{\boldsymbol{W}}$ scales with SB, where B is the batch of samples used in the gradient calculations. As noted in section 4.1, backpropagation is more efficient than parameter-shift methods because it can calculate derivatives in parallel. In contrast, parameter-shift methods require two or more shifted expectation values to calculate the gradient with respect to a single parameter. P generally increases with the depth of the circuit, so for circuits with significant depth, the quantum resources needed to train the neural network are insignificant compared to training the circuit parameters.

Training these algorithms instead of bare VQAs seems to be a more efficient use of the quantum resources due to the computational scaling of the gradient calculations. With the idea that the weights of the neural network alter the cost landscape with respect to the gate parameters, this thesis investigates whether a simple feedforward neural network can aid VQAs in finding better solutions. The following subsection considers the ESCAPE algorithm.

7.1 ESCAPE-algorithm

The ESCAPE algorithm is utilized when the optimization of the VQA encounters a local minimum. In essence, the algorithm acts as a quantum version of basin-hopping [60] where parameters are perturbed until a new minimum is found. However, in contrast to the traditional basin hopping, the parameters are here kept fixed while the cost landscape around the parameters is altered. In what follows, the precise steps of the algorithm are presented from the paper of Riveradean et al. [4]. Along with each step, a short description of the step is provided. For the sake of generalizability, the entire neural network will be referred to as a function $f_{NN,\boldsymbol{W}}$

where $\boldsymbol{W}$ refers to the trainable parameters of the neural network. It takes as input the measurement data of the quantum computer and outputs a string from which a cost can be evaluated.

1. Initialize $\boldsymbol{\theta}$ and set $f_{NN} = \mathbb{1}$
2. Update $\boldsymbol{\theta}$ via a gradient descent until convergence to a local minimum.

   · *So far the algorithm trains without the influence of the neural network, hence $f_{NN} = \mathbb{1}$. As it reaches the minimum, the VQA is stuck in a potentially suboptimal minimum.*

3. Update $\boldsymbol{W}$ via a gradient descent procedure for M steps and define $\boldsymbol{W_0}$ as the weight matrices after the last step.

   · *The circuit is now a fixed sample generator used to train the weights $\boldsymbol{W}$ of the neural network.*

4. For $t = 1, \ldots, T$:

   (a) Set $f_{NN,\boldsymbol{W}} = (1 - g(t)) f_{NN,\boldsymbol{W_0}} + g(t)\mathbb{1}$ where $g(0) = 0$ and $g(T) = 1$.

   (b) Perform a gradient descent update step of $\boldsymbol{\theta}$,

   · *In this part of the algorithm, the circuit parameters $\boldsymbol{\theta}$ are again optimized, but the cost landscape gradually changes from the altered cost landscape by the neural network to the original cost landscape due to the properties of $g(t)$. Through this continuous deformation of landscape, the $\boldsymbol{\theta}$ parameters are potentially perturbed away from the current local minimum of step 2.*

5. Update $\boldsymbol{\theta}$ via gradient descent until convergence to a (potentially different) local minimum.

   · *Now the cost-landscape is reverted back to the original problem ($f_{NN} = \mathbb{1}$) and the local minimum potentially escaped. The optimization procedure of this step solves the original VQA problem.*

6. Compare energies in step 2) and step 5). Pick the solution with the lowest energy.

A figure describing the ESCAPE procedure is found in figure 25 where the neural network alters the landscape, then gradually relaxes back to the original landscape again. This gradual perturbation causes the parameter vector to reach one of the global minima of the landscape.

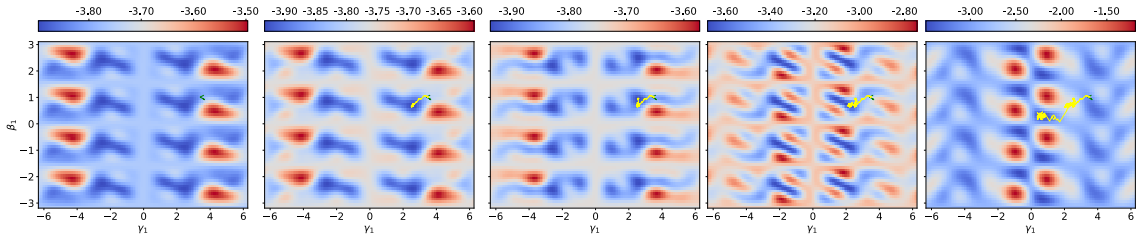


Figure 25: These subplots are meant to exemplify step 4 of the ESCAPE procedure where the neural network alters the cost landscape with respect to the QAOA parameters. These subplots show the cost landscape as a function of $(\gamma_1^{p=1}, \beta_1^{p=1})$ at various times t with $g(t) = t/T$ and $T = 350$ for a five node weighted fully connected graph. The yellow line show the optimization path taken during step 4 of ESCAPE. See repository [61] for an animation of this transition.

7.2 Details on the implementation

Similar to the original paper, the ESCAPE algorithm was implemented using the python library Pennylane. Pennylane provides qubit simulators where the entire quantum circuit is end-to-end differentiable and uses methods from traditional machine learning to speed up the gradient calculations. Their simulator devices construct a computational graph upon a forward pass through the quantum circuit. Through the computational graph, the gradient with respect to the free parameters in the VQA ansatz can be calculated using back-propagation. This gradient could, in principle, be calculated by utilizing the parameter-shift rule (which is also built into Pennylane); however, it requires significantly longer computations as the ansatz depth increases. In essence, these simulators are statevector simulators capable of calculating the exact outcome of the quantum computer up to floating-point precision. This, in principle, means that the same set of parameters $\boldsymbol{\theta}$ that define an ansatz will give rise to one specific output. From this point forward, this simulator device will be called the "ideal simulator." Pennylane also allows for a shot-based simulator that outputs a fixed amount of measurements in the computational basis. This simulator device will be called the shot-based simulator. The choice of Pennylane over other frameworks such as Qiskit was made because gradient computations are more intuitively built into this programming framework. Additionally, the gradient computations are faster because the backend is built up with gradient calculation in mind. This framework flows seamlessly into other machine learning libraries such as PyTorch.

Particularly for the ESCAPE algorithm, one can use the ideal simulators provided by Pennylane for steps 2 and 5 to reach deep parts of the cost landscape. This might be infeasible when using a shot-based approach since the stochastic output hinders it from reaching deeper parts of the landscape. However, step 4 of the algorithm might prove problematic depending on the problem's size. Rivereadean et al.' s [4] workaround to incorporate the analytical gradient framework was to let the quantum device return a list of the probabilities of measuring each possible basis state $\{\boldsymbol{x}\}$. Through these probabilities, the cost function of the problem can be readily calculated classically by inserting these probabilities into the expression given earlier in equation 113. Since these probabilities, in principle, are deterministic (one

specific set of parameters $\boldsymbol{\theta}$ would give rise to one specific probability distribution), the gradient with respect to the VQA parameters $\boldsymbol{\theta}$ can be calculated using

$$\frac{\partial p_i}{\partial \theta} = \frac{\partial \langle \psi(\theta)|i\rangle\langle i|\psi(\theta)\rangle}{\partial \theta} \tag{120}$$

and hence the parameters can be optimized. However, this is a major computational bottleneck. Notably, as the qubit count increases, the number of possible basis states $\{\boldsymbol{x}\}$ increases exponentially as 2^N. Hence, if one wishes to increase the problem sizes beyond $N \sim 16$ qubits, a different approach is needed (this was the highest qubit-count simulated in the original paper).

At graph instances with twelve nodes, the ESCAPE procedure using the ideal simulator described above takes a modest amount of computational time. Therefore, the ESCAPE routine in its original form is tested for graph instances up to twelve nodes to reproduce the findings of the original paper. As the number of graph nodes increases, a shift is made to move away from the ideal simulator and instead utilize a more realistic shot-based quantum computer. In principle, one could continue to use the ESCAPE procedure in its intended form by utilizing the parameter shift rule to get a stochastic estimate of the analytical gradient. However, in the hopes of using fewer function evaluations throughout the procedure, a change from the original approach is made to use gradient-free methods.

Two particular gradient-free methods were tested; Nelder-Mead (as implemented in SCIPY [57]) and the Simultaneous Perturbation Stochastic Approximation (SPSA). Rather than taking a single gradient-based step in step 4b of ESCAPE, one instead takes multiple gradient-free steps to approximate the single gradient-based step. This altered approach changes the cost function in step 4 in two ways. Firstly, the sum over $\{\boldsymbol{x}\}$ is now only sampled from the quantum circuit instead of the entire set of possible bitstring combinations as was done in the ideal approach, hence circumventing the computational bottleneck. Secondly, $p(\boldsymbol{x}|\boldsymbol{\theta})$ is instead estimated based on the samples returned from the circuit instead of being the exact probabilities of measuring every bitstring. The number of shots S is set to a fixed number of 10 000 throughout the shot-based simulations. This causes uncertainty in the cost calculations to fluctuate around 0.01, as explained in Appendix C. This choice introduces noise into the system, but not too much to ultimately hinder optimization. To demonstrate that escaping a local minimum using gradient-free steps is feasible, the following section considers a toy example to examine the mechanisms underpinning this procedure.

7.3 Toy Example

A toy model was constructed and studied in some detail to understand the underlying mechanism that allows for the hopping from one minimum to another. In this example, a simple "quantum computer" is considered that outputs the parameters (γ, β), i.e.

$$(\gamma, \beta) \longrightarrow \text{QC} \longrightarrow (\gamma, \beta). \tag{121}$$

In other words, the system essentially performs an identity transformation instead of the QAOA transformation with its binary bit-string outcome. This is a massive simplification of the original system; however, one may extract valuable insights using simpler systems. This system is then attached to a 2×2 neural network which processes this outcome and outputs a $\mathbb{R}^2$ vector. This vector is the input to some cost function C, which outputs a real number $\mathbb{R}$. Schematically the toy example considered in this problem is given as

$$(\gamma, \beta) \longrightarrow \text{QC} \longrightarrow (\gamma, \beta) \longrightarrow \text{NN} \longrightarrow \text{Cost}. \tag{122}$$

The cost landscape depends on the original system parameters (γ, β) and the neural network parameters $\boldsymbol{W}$. Seen from a different perspective, the neural network can alter the cost landscape when plotted as a function of (γ, β). This section aims to demonstrate that it is possible to escape a local minimum using a neural network and what mechanisms in the neural network cause the escape.

This toy example considers a neural network without any activation functions. Therefore, only a linear transformation from the input to the output is performed. The cost function considered here is given by a sum of two-dimensional normal distributed functions

$$\text{Cost}(x, y) = \sum_i \alpha_i \cdot \mathcal{N}(x, y | \boldsymbol{\mu_i}, \sigma_i^2) \tag{123}$$

By altering the value of α_i and locating these circles at different points in parameter space, one creates a custom cost landscape with various local minima. In these examples, a landscape with three Gaussians was considered with the following parameters:

i	α_i	$\boldsymbol{\mu_i}$	σ_i^2
1	-1	$[-20, 10]$	10
2	-1.5	$[-12, -3]$	10
3	-2	$[6, -12]$	10

The procedure goes as follows. One chooses a point relatively close to the local minimum, $\boldsymbol{x_1}$, with the highest cost value. Then train the neural network parameters by using the cost function in eq. 123 and the current parameter-values $\boldsymbol{x_1}$ in the local minimum; $C(\text{NN}(\boldsymbol{x_0}))$. Then, for T steps, gradually turn off the trained neural network parameters, $\boldsymbol{W_0}$, and change them with the identity matrix by the following:

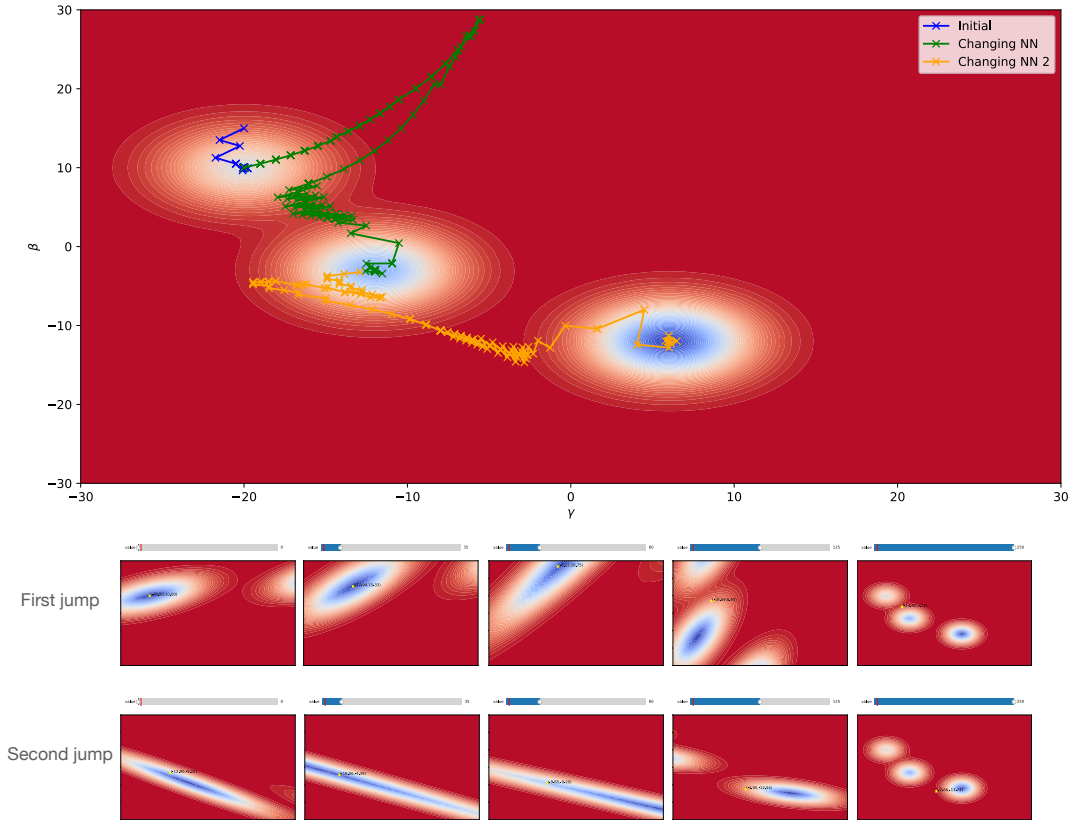


Figure 26: A plot of the cost landscape and the two consecutive jumps performed by the procedure using a simple feedforward neural network without any hidden layers or activation functions. The blue line is the initial optimization when starting from a point close to the worst minimum. The green line shows the optimization path taken by the procedure when jumping from minimum μ_1 to μ_2, while the yellow line shows the trajectory taken to jump from μ_2 to μ_3. The two lower rows of the figure show snapshots of the first and second jump respectively in various parts of the optimization procedure to highlight how the neural network alters the landscape. See repository [61] for an animation of these jumps.

$$\boldsymbol{W} = \left(1 - \frac{t}{T}\right) \boldsymbol{W_0} + \frac{t}{T} \mathbb{1} \tag{124}$$

During each of these steps, update the parameter values $\boldsymbol{x}$ by performing a maximum of 5 iterations of a gradient-free optimizer such as Nelder-Mead or a gradient-based optimizer such as BFGS. A limitation on the number of iterations for these gradient-free procedures was put in place to limit the number of function evaluations that can be taken at each t since the original ESCAPE procedure would only perform a single gradient-based step at each t. Simulations were run with this setup to see whether successive escapes were possible from the minimum located close to $\boldsymbol{\mu_1}$ to $\boldsymbol{\mu_2}$ then from $\boldsymbol{\mu_2}$ to $\boldsymbol{\mu_3}$.

Figure 26 shows an example of when the procedure successfully jumps to better minima twice. In order to understand the dynamics of what happens during these

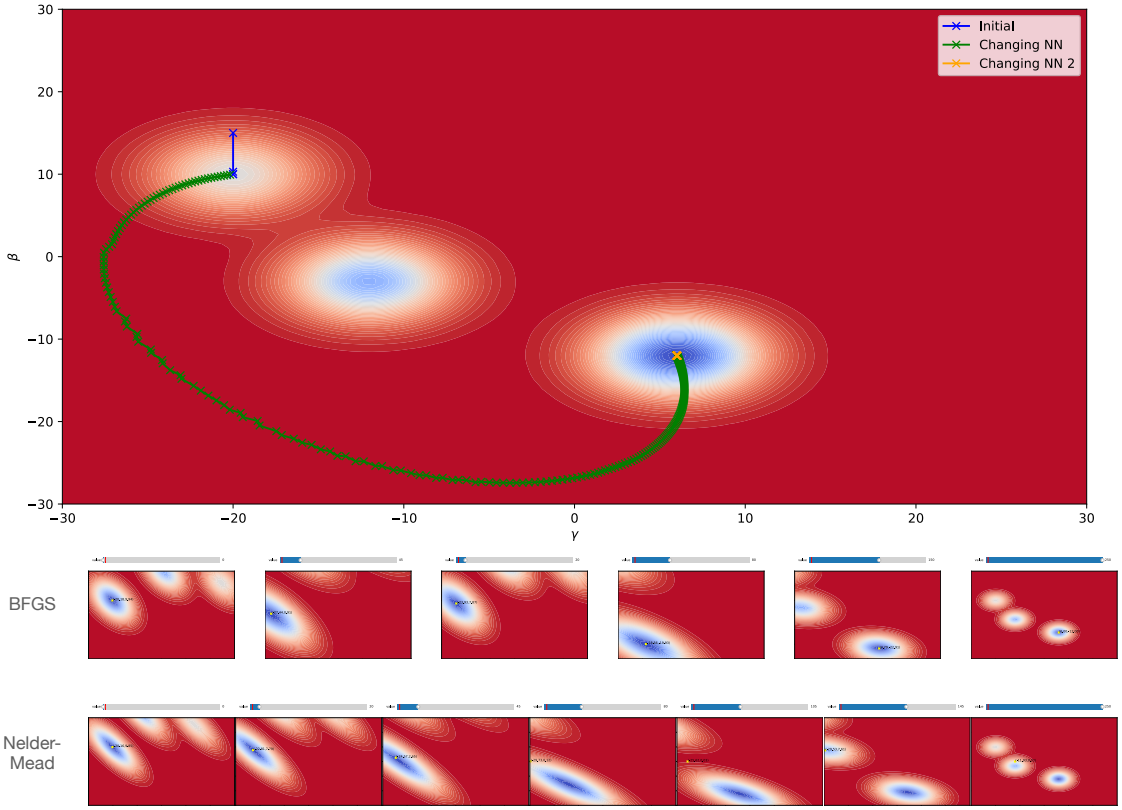


Figure 27: A simulation where the method was able to map the first minimum to the global minimum using a single jump where at each time t a maximum of 2 BFGS steps were used. The first row shows snapshots of the optimization at various t for this direct jump from the first minimum to the global one. This sequence shows that the global minimum of the cost landscape is mapped to the original minimum x_0, which is then relaxed to the global minimum of the original cost landscape. A similar procedure was also performed using steps of Nelder-Mead instead of BFGS, where the neural network is able to map the original minimum to the global minimum. However, during relaxation, the optimization procedure escapes the attractive basin of the global minimum and instead relaxes to the second-best minimum x_2.

successful jumps, the landscape at various times t are plotted along with the current values of (γ, β) at those points in the lower two rows of the same figure.

Due to the simplistic nature of the neural network, it is relatively easy to analyze its behavior. Firstly, note that this neural network is limited to rotation and stretching/shrinking of the cost landscape due to the absence of an activation function. During a successful escape from a local minimum, the neural network parameters are chosen such that it can map the parameters in the minimum, $\boldsymbol{x_0}$, to a minimum of lower cost. The first snapshot of the first jump in figure 26 illustrates this where the minimum point $\boldsymbol{x_0}$ overlaps with the light-blue region. At the end of the transformation, this region is continuously transformed into the second-deepest minimum of the original cost landscape. As one takes optimization steps at each t, the parameters tend to stay within this local minimum at each time t. The second jump also exhibits the same trend where the cost landscape is initially stretched

such that the global minimum overlaps with $\boldsymbol{x_1}$ then continuously transformed back into the actual position of the global minima of the original cost landscape.

A direct jump to the global minimum is also possible in this toy example. Using the same setup as before, one may encounter a situation as presented in figure 27 where the neural network can map $\boldsymbol{x_0}$ to the global minimum of the landscape. In this simulation, the steps in the changing landscape were performed using the gradient-based minimization procedure BFGS. Snapshots of this simulation are presented in the first row of the same figure. On the other hand, the second row presents a similar situation where the neural network was able to map the original minimum to the global one. However, using the Nelder-Mead optimizer instead of BFGS caused the procedure to perturb out of the attractive basin during relaxation. It, therefore, seems that BFGS requires fewer steps to remain in the attractive basin due to the added gradient information. Suppose one were to use gradient-free optimizers like Nelder-Mead. In that case, one has to either increase the maximum number of gradient-free steps at each t so that the parameter vector remains within the attractive basin at each step or increase the relaxation time T so that the cost landscape is less severely changed at each step.

Another point of failure for this method is the inability of the neural network to map the starting local minimum to a better minimum. The first jump to the second minimum is relatively consistent in this example. However, the second jump to the global minimum is less so and relies heavily on if the initial random parameters of the neural network could be trained to perform the necessary mapping that results in the global minimum. Based on these observations, the distance between the minima seems to be the primary cause of this issue.

As mentioned earlier, this toy-example landscape is not indicative of an actual cost landscape for QAOA applied on Max-Cut for $p = 1$ parameters. Instead, a relatively barren landscape was created to examine certain limitations of this procedure in this extreme case. Additionally, since the parameters in QAOA are angles, they are at least 2π periodic; hence the landscape is not as far-reaching as it is in this example. Therefore, this gradient-free version of ESCAPE seems promising for the problem of QAOA on MaxCut.

7.4 Toy Example: Overparametrization and activation functions

This section extends the results found from the precious section slightly further by introducing more nodes, layers, and different activation functions to the neural network to see if they aid in training. Various setups with different hidden layers and the number of nodes within the hidden layers were tested. This method is therefore not limited to only stretching and rotating the landscape but is instead able to perform nonlinear transformations of the landscape to potentially map the initial point $\boldsymbol{x_0}$ to a better minimum. For each setup configuration, 100 simulations were run where one starts the procedure initially close to the worst minimum, then attempts two jumps where the neural network is trained then gradually relaxed back

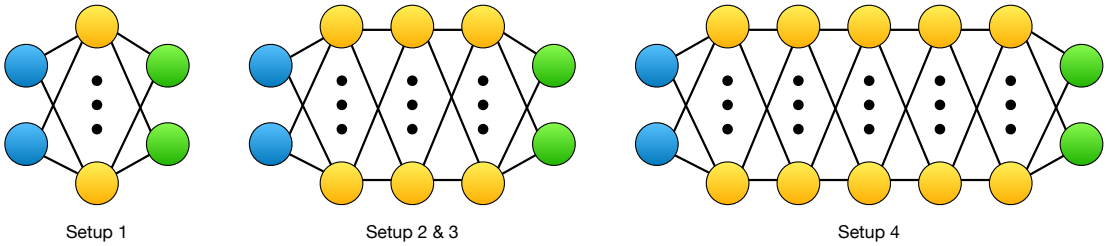


Figure 28: Setup 1 consists of one hidden layer with the ReLU(x) activation function, Setup 2 consists of three hidden layers with the $\tanh(x)$ activation function, Setup 3 consists of three hidden layers with the ReLU(x) activation function and Setup 4 consists of five hidden layers with ReLU(x) activation. Within each of the setups, the procedure was tested with various numbers of nodes, and the results are shown in table 3.

to the original cost landscape again. For all these simulations, the neural network was trained using the Adam optimizer with an initial learning rate of 0.01 for 250 optimization steps. The relaxation back to the original landscape is performed using the BFGS optimizer for a maximum of 5 steps. In these simulations, the only thing that is allowed to change is the random weights that initialize the neural network. Although one of the setups performs better than the naive method of no hidden layers, these methods still struggle with the same problems that the previous method encountered, namely that if the distance between minima is too large, the method fails. Considering that the landscape is mostly barren, and hence little gradient information is available, one might anticipate such outcomes.

Number of nodes	Setup 1	Setup 2	Setup 3	Setup 4
5	14	2	7	6
10	17	2	5	4
20	11	0	5	0
100	14	0	3	0

Table 3: A table showing the number of attempts out of 100 that the method found the lowest minima in the example landscape. Within each of these setups as described in figure 28, the number of nodes in the hidden layer is given in each row of the table. The number of successful jumps in the case of no hidden layer and bias terms is 10. From these results it is evident that Setup 1 is the best performer.

There are two key takeaways from these tests. Firstly, using activation functions and increasing the number of trainable parameters in the neural network seems to aid in searching larger parts of the optimization landscape as there are more degrees of freedom to optimize with respect to the over parametrized neural network. However, this is not a general result as table 3 shows where the best performing setup is one where one has a single hidden layer with ten nodes in it with the ReLU activation function. Rather surprisingly, it seems that merely increasing the number of nodes in the network does not aid in searching a larger part of the landscape as one might anticipate. It is uncertain why this is. One possible reason for the lowered performance with increasing complexity might be that the input and

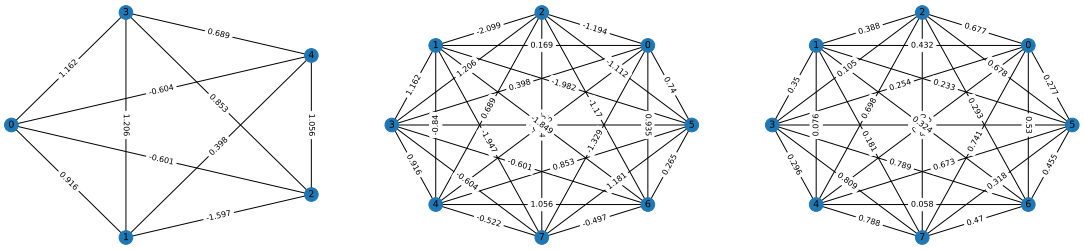
output dimensions are only 2, and hence merely increasing the number of trainable parameters in the network inhibits training as one is easily capable of finding a parameter configuration that maps to the closest minimum rather than searching the landscape.

Secondly, the same shortcomings that were found in the simpler examples still remain in these over-parametrized problems as well. Notably that the procedure is heavily dependent on whether the neural network is capable of mapping a better minimum to the initial starting parameter-value $\boldsymbol{x_0}$. If this is the case, a gradual relaxation towards the original cost landscape will lead to a better minimum in the landscape. If this is not the case, a better minimum is not found and the relaxation remains in the originally found local optimum. Hence for the difficult problems considered here, the procedure is heavily dependent on the initial values of the neural network weights. The primary limiting factor of jumping towards better minima seems to be the distance between the minima, and also how deep the original minimum is.

7.5 Reproduction and extension of ESCAPE

As indicated in the previous section, the method of perturbing a point away from a local minimum by altering the landscape can work. This section first implements the method presented by Riveradean et al. [4] where each optimization step in part 4 of the algorithm is made using gradients on the ideal simulator. The algorithm was tested on various graphs of a similar type as the ones presented in the paper of Riveradean et al. [4], namely

- Graph A: A five node fully connected graph with weights sampled from a normal distribution with $\mu = 0$ and $\sigma^2 = 1$
- Graph B: An eight node fully connected graph with weights sampled randomly between $[0, 1]$
- Graph C: The same eight node fully connected graph but with weights sampled from a normal distribution with $\mu = 0$ and $\sigma^2 = 1$



(a) Graph instance A (b) Graph instance B (c) Graph instance C

Figure 29: The various graph instances that are considered in this section

Note that the graph weights are allowed to be negative in graphs A and C; hence, some graph partitions may have negative costs. For the approximation ratio to remain meaningful for these graph instances, it is changed slightly to the following:

$$r = \frac{\langle C(\boldsymbol{\gamma}, \boldsymbol{\beta}) \rangle - C_{min}}{C_{max} - C_{min}} \tag{125}$$

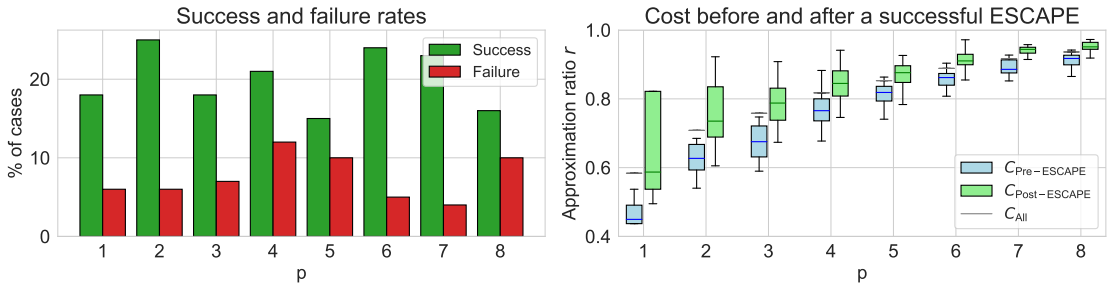
With this change, the approximation ratio will remain within the interval $[0, 1]$ even if the mean cost is negative. The original paper only considered $g(t) = \Theta(t - 150)$ where Θ is the Heaviside function. This thesis also considers a gradual relaxation out of the cost landscape using $g(t) = t/T$ to see if that provides any difference in performance.

For each QAOA depth p, 100 simulations were run where the initial parameters were sampled randomly between $[0, 2\pi]$. The results of these simulations will be presented in a similar format to figure 30. The leftmost subplots show the number of successful and failed escapes that were performed over the 100 random initializations. The rightmost subplots show the change in energy for those initializations where the ESCAPE algorithm successfully found a better minimum relative to the initial minimum from step 2. Riveradean et al. [4] determine a successful escape based on the condition $\mathcal{C}(\boldsymbol{\theta}_{\text{QAOA}}) - \mathcal{C}(\boldsymbol{\theta}^*) > 0.1$ where $\boldsymbol{\theta}_{\text{QAOA}}$ are the initial parameters at the local minimum of step 2 while $\boldsymbol{\theta}^*$ are the final parameters found after the ESCAPE procedure. $\mathcal{C}$ is the cost function that defines the problem being solved (MaxCut cost function throughout this thesis). A cost difference of -0.1 or less categorizes a failed jump, i.e., the ESCAPE procedure produced a worse final minimum by the same condition. For the remaining cases where the cost difference is between -0.1 and 0.1 it is assumed that the ESCAPE procedure finds the same minimum. Since an ideal simulator is used throughout this section, using this condition as a measure of success is valid since the cost can be calculated to floating-point precision.

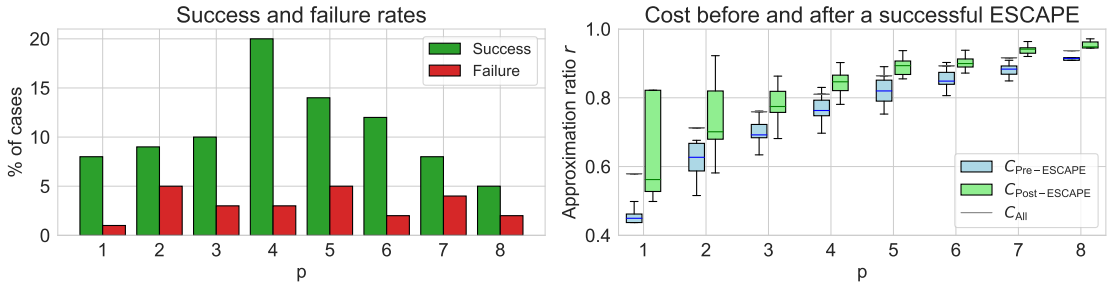
The various graph instances highlight certain aspects of the gradient-based ESCAPE routine. Firstly, these graphs are used to highlight the difference between using $g(t) = \Theta(t - T)$ instead of using $g(t) = \frac{t}{T}$. Graph A also highlights the observation that over-training the neural network may hinder the performance, a result found for both versions of $g(t)$ that were tested. Lastly, graph instances B and C show that when the graph instances have several cuts of negative value, the performance of the procedure shows a trend of increasing successes with increasing depth p. Riveradean et al. [4] found this trend for all their graph instances. The observations from these graph instances seem to suggest that this trend is graph instance dependent and also dependent on the number of initial steps performed at step 2 of the ESCAPE routine.

7.5.1 Graph instance A

Solving on identical graphs of type A, a simulation involving $M = 50$ neural network optimization steps is compared to a simulation involving $M = 200$ steps. The neural network optimization is performed using stochastic gradient descent (SGD) with a learning rate of 0.05. The results of $M = 50$ ($M = 200$) are shown in figure 30a (figure 30b). When the neural network is overtrained, most runs find the same optimum that it started in, as indicated by the lowered success and failure rates



(a) Results using 50 neural network training steps

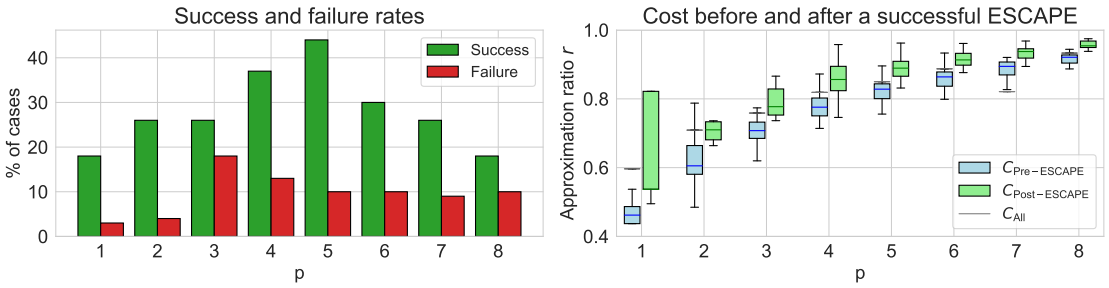


(b) Results using 200 neural network training steps

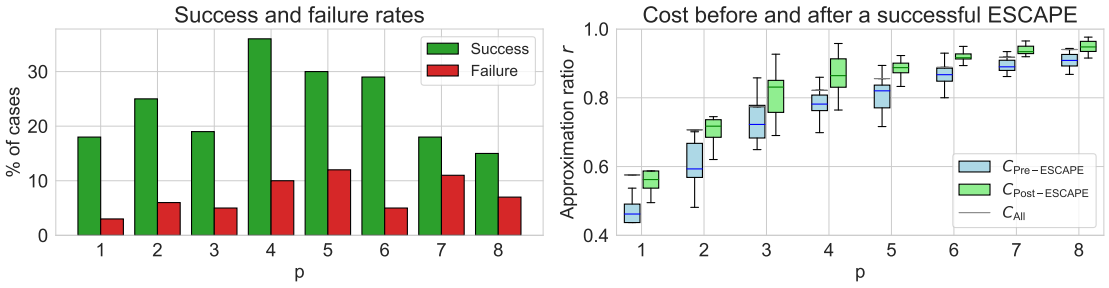
Figure 30: The statistics of the ESCAPE procedure applied to graph instance A. Within these subplots, the left subplot shows the number of cases out of 100 initializations where the procedure succeeded or failed using the success criteria as presented in the text. Given that a successful escape has been performed, the right subplot shows the cost before (blue) and after (green) the escape as whisker plots. The average cost over all 100 initializations is also plotted, given by the grey lines C_{All}. Figure 30a (figure 30b) shows the performance when $M = 50$ (200) neural network steps are taken during training. At step 4 of ECAPE, $g(t) = \Theta(t - 150)$ is used. As the plots show, the number of successful jumps seems to increase when the neural network is not overtrained; however, the number of failures also seems to increase.

of figure 30b. It would seem that the parameter vector cannot be perturbed away from the local minimum it is stuck in when the landscape is overtrained. Note that the training landscape in these simulations are static due to $g(t) = \Theta(t - 150)$. During step 3 of ESCAPE, $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ remain fixed, and the quantum computer acts like a sample generator that outputs bitstrings that are input to the neural network. As mentioned earlier, the neural network maps an input binary string into an output one with a better cost. Therefore, the neural network deepens the cost landscape around the originally found $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ to a value that has an approximation ratio closer to 1. This deepening of the cost landscape can be seen in figure 25. Since the cost cannot improve any further, the QAOA parameters are perturbed little in this altered landscape during step 4 of the ESCAPE procedure.

Using the same graph, another set of simulations using $g(t) = t/T$ with $T = 350$ was run and compared with the previously mentioned results. Similar to before, two simulations were run; one with $M = 50$ training steps (figure 31a) and another with $M = 200$ (figure 31b). As shown in figure 31, the overall performance seems to have improved when using $g(t) = t/T$ instead of using $g(t) = \Theta(t - 150)$. This



(a) Results using 50 neural network training steps



(b) Results using 200 neural network training steps

Figure 31: The same type of plots as the previous figure, however, with the change that $g(t) = \frac{t}{T}$ with T set to 350.

seems to indicate that training in a dynamic landscape is better than training in a fixed landscape. Using the insights gained from the toy example, this improvement in performance can be attributed to the finding that dynamically changing the cost landscape forces the parameter-vector to perturb away from the originally found minimum, thus potentially improving the found cost. From the toy-example it is also evident that even if the original parameters are potentially mapped to better cost values, the jump itself may fail due to the parameter-vector jumping out of the attractive basin created by the neural network. These results also seem to show that using an over-trained neural network decreases the success rate, similar to what was found when using the Heaviside function.

For all the four variations considered here, it is found that the method succeeds for those cases where the initial QAOA cost is worse than the initial cost that is found on average from the 100 simulations. This can be seen from the right subplots of these figures, where the mean over all 100 simulations (C_{All}) has a higher cost than the mean of $C_{\text{Pre-ESCAPE}}$ indicated by the blue lines within the blue boxplots. This reiterates that the algorithm is more of a corrective method meant to find slightly better solutions when a local minimum is found rather than an approach meant to find the optimal minimum of the landscape, such as the INTERP heuristic.

The choice in hyperparameters was meant to mimic the ones of the original paper of Riveradean et al. [4]. During step 2 of the algorithm, the optimization using Adam was performed for 200 steps to ensure convergence. However, one specific pattern was not reproduced with these simulations. Namely, Riveradean et al. [4] claim that the number of successful escapes increase with QAOA depth p, and find that the number of successful escapes hovers around $30 - 40\%$ for graph instance A when

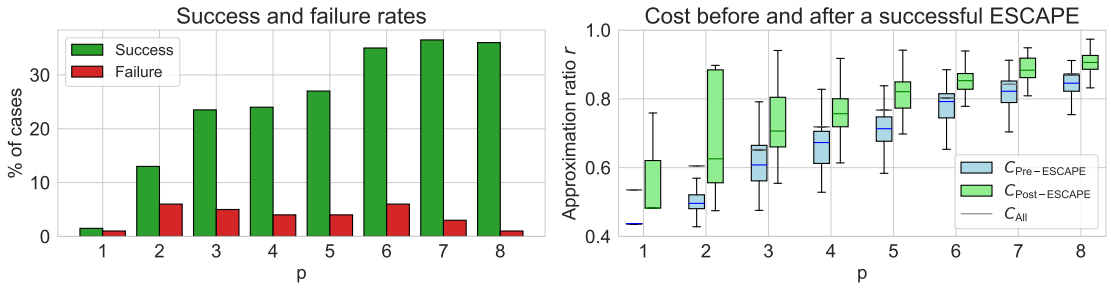


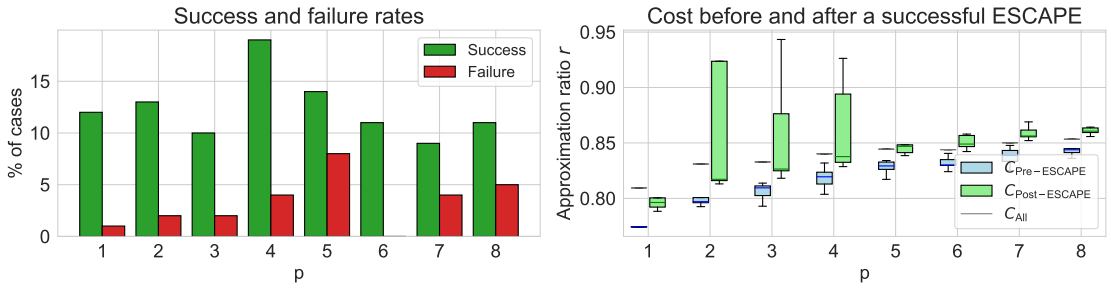
Figure 32: A figure showing the findings that Riveradean et al. [4] highlight. Namely, as the QAOA depth p increases, so does the number of successful escapes. Note that the number of steps taken during the initial QAOA optimization is 50 instead of 200.

using $g(t) = \Theta(t-150)$. Evidently, this was not the case for the simulations that were performed here, as no particular pattern was found. However, when changing the number of initial optimization steps during step 2 of ESCAPE from 200 down to 50, these patterns that Riveradean et al. [4] find start to emerge as shown in figure 32. Namely, the number of successful escapes increases with increasing p. It, therefore, seems that the pattern that Riveradean et al. [4] find is mostly a consequence of terminating the initial optimization before it has completely converged to the local minimum. One can also see the effect of this from the right subplot of figure 32 where the blue boxplots and C_{All} have lower approximation ratios than for figures 30 and 31. Since 50 steps are not enough for convergence, particularly for higher depths p, the additional optimization steps during steps 4 and 5 of the ESCAPE procedure will most likely cause an improvement in cost of at least 0.1 even if the parameter-vector is not perturbed significantly away from the initially found minimum of step 2.

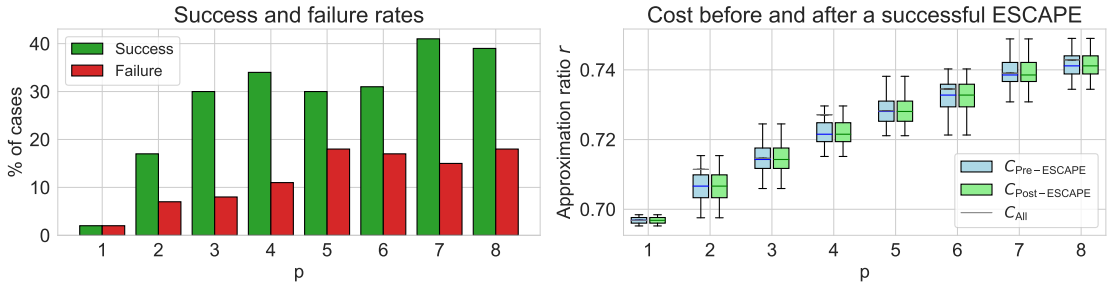
The most deviation from this pattern is found at the higher depths p of around 7 and 8, where the number of successes decreases as shown in figures 30 and 31. An explanation for this deviation is that the graph instance being solved is relatively simple. Therefore, using a deep QAOA ansatz to solve the problem would result in high-quality minima during the initial optimization, as indicated by C_{All}. As a result, few minima are available in the landscape that can create a cost-improvement of at least 0.1 as required by the success criteria.

7.5.2 Graph instance B and C

One distinct difference between graph instances B and C is that graph instance C may have negative weights. For graph instance C, 232 out of all possible 264 partitions have negative costs associated with them; hence most partitions of the graphs have subpar cuts. It is also worth noting that only 6 out of 32 possible partitions of graph A had negative costs. Two sets of similar simulations were run on each graph instance to see whether this caused any differences in the number of successful escapes. Namely $M = 50$, $g(t) = \Theta(t - 250)$ and 300 initial optimization steps of the Adam optimizer is taken at step 2 of ESCAPE. The initial learning rate η is 0.1.



(a) Results using graph B.



(b) Results using graph C.

Figure 33: The performance of the gradient based ESCAPE routine applied to graph instances B and C. In these simulations $M = 50$, $g(t) = \Theta(t - 250)$ and the initial learning rate of Adam is $\eta = 0.1$ with 300 initial optimization steps.

As figure 33 shows, the rate of successful escapes increases when using graphs of type C over type B. Additionally, the number of successful escapes increases with increasing QAOA depth p for graph C. As this example shows, the trends that Riveradean et al. [4] find seem graph dependent as well. It is difficult to pinpoint what causes the differences in performance when only the graph weights are changed. Considering that graphs A and B have relatively high approximation ratios in contrast to graph C, it might be that those two graph instances are easier to solve using the QAOA ansatz. Consequently, there are few local minima that the procedure can escape into, allowing for a cost improvement of 0.1. Additionally, the cost landscape has a different nature for graph instance C compared to graph instance B. Note that the cost function during optimization is $-\langle\psi(\boldsymbol{\gamma},\boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle$, i.e. a weighted sum of all partitions of the graph. Since most partitions are negative, the cost landscape is primarily positive, except for regions where the good partitions are heavily weighted. In other words, the cost can undergo a sign-change during the optimization to find regions of good cost. This span in cost in the landscape may explain why the performance is higher overall for graph instance C over B, particularly for higher depths p.

This concludes the reproduction part of the ESCAPE procedure. As found in this section, the trend of increasing successful escapes with increasing p is not a general finding for this algorithm, especially for the smaller graph instances. This can be attributed to the notion that it is relatively easy to solve these graph instances; hence using QAOA with randomized initializations often find decent solutions on their own. The procedure's success depends heavily on the number of initial optimization steps at step 2 and the graph instance considered. From this point forward, the novel

shot-based approach presented in the implementation section will be used.

7.6 ESCAPE with sampling noise

This section tests the ESCAPE routine using a shot-based quantum computer. The quantum computer is still a simulator; however, its results are now a finite set of samples taken over the resulting probability distribution over all possible bit strings at the end of the quantum computation. This contrasts with the previously used ideal simulator, where the entire procedure was deterministic; a set of inputs that define the system, $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ would give a single cost output up to floating-point precision. In other words, the number of shots taken in the ideal simulator is effectively infinite.

The reason for using a shot-based approach instead of the deterministic approach as described in section 7.1 is because it is more realistic to do so. All current quantum computers give stochastic bitstring output. Hence, it is more interesting to see how using a neural network would aid in a more realistic implementation of the procedure. Additionally, as mentioned earlier, the gradient-based approach inevitably does not scale to a higher number of qubits due to the exponential scaling in the $p(\boldsymbol{x}|\boldsymbol{\theta})$ term of the cost function.

Various optimization procedures can be used in this version of the algorithm, both gradient-based and gradient-free methods. Usually, gradient-based methods are shown to work reasonably well on most problems; however, this comes at the cost of a higher number of function evaluations. These function evaluations are costly as the problem instances grow in size. For instance, a central finite difference method would require two function evaluations per parameter, resulting in $4p$ function evaluations per gradient calculation for the QAOA ansatz. As mentioned in section 4.1, the scaling is even worse for a general parameter shift rule approach as it would scale like $2(2R + 1)p$ for the QAOA ansatz. These issues motivate using gradient-free approaches to reduce the number of needed function evaluations while optimizing.

Two gradient-free variations of the ESCAPE procedure were implemented and tested out. These procedures differ from the original procedure primarily in steps 2, 4, and 5. In steps 2 and 5, a gradient-free optimization procedure is used in the original cost landscape instead of a gradient-based one to find a local minimum. In step 4, one takes a fixed number of gradient-free optimization steps to approximate the single gradient-based update step of the original procedure. The first method uses the Simultaneous Perturbation Stochastic Approximation (SPSA) as the gradient-free optimizer, whereas the second uses Nelder-Mead. These particular gradient-free optimizers were used because they are less prone to noise and are easily implementable. Particularly with SPSA, the number of function evaluations at each optimization step is always two, as noted in section 2.4.2, regardless of the dimension of the parameter vector.

The shot-based ESCAPE procedure was implemented and tested for two different graph instances,

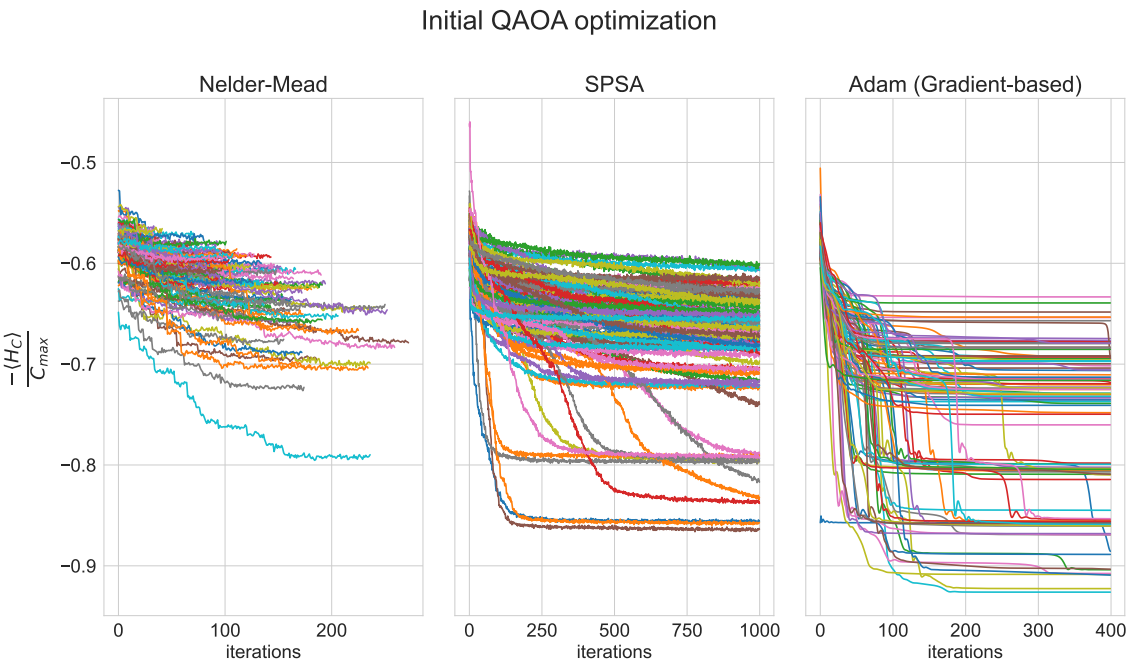


Figure 34: The cost plotted against the steps of the initial optimization procedure at QAOA depth $p = 8$. Note that the optimization is performed on $-\langle\psi(\boldsymbol{\gamma},\boldsymbol{\beta})|H_C|\psi(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle$, thus the approximation ratio r can be extracted by multiplying the above graphs with -1. Three different optimizers were used: Nelder-Mead, SPSA, and Adam. For each case, 100 different initial parameters were chosen and optimized. The Nelder-Mead and SPSA consider sample-based function evaluations while the simulation involving Adam uses an ideal quantum computer with analytical gradients, The convergence criteria for Nelder-Mead was set to y-tol $= 0.1$, x-tol $= 0.02$. For SPSA, the hyperparameters (a, c) were set to $(0.2, 0.15)$ respectively and the optimization procedure was performed for 1000 steps. For Adam, the initial learning rate was set to $\eta = 0.05$ and 400 iterations were performed.

- Graph D: A twelve node w3R graph with weights sampled randomly between [0,1]
- Graph E: A sixteen node w3R graph with weights sampled randomly between [0,1]

Graph D is used primarily as a benchmark for the procedure where the shot-based version of ESCAPE is compared to the original gradient-based ESCAPE procedure on an ideal quantum simulator. For this graph instance, the ESCAPE algorithm was implemented with $T = 800$ and $g(t) = \frac{t}{T}$. For the gradient-free ESCAPE, three gradient-free steps were performed at each t during step 4 of the procedure. This section presents some findings when moving from an ideal quantum computer to a shot-based one.

The first finding is that navigating the cost landscape when sampling noise is present is difficult, as evident from figure 34 where several parameter initializations converge to worse cost values than in the no-noise setting. Nelder-Mead is the worst performer, while the gradient-based method Adam is the best performer, as evident by

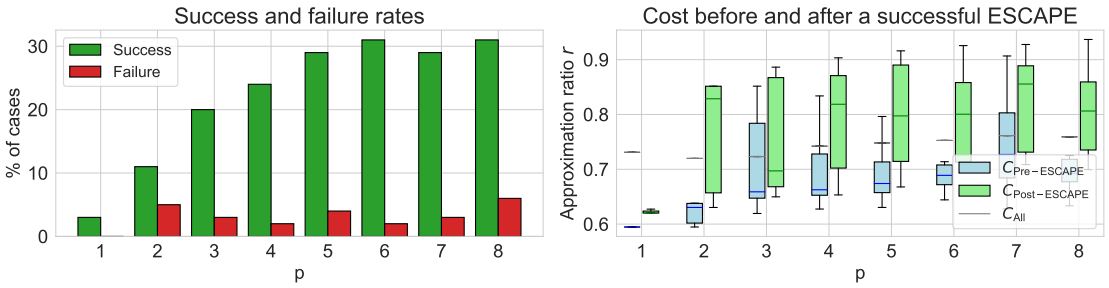
the number of initializations that reach deeper parts of the cost landscape. There are a couple of reasons that may explain the lack of performance in the gradient-free methods. The first is simply that the method converged to a local minimum with a high cost. Considering that several optimizations using Adam also converge to minima with high cost, one would expect the same from gradient-free methods too. Additionally, since Adam uses additional information in the form of gradients to navigate the cost landscape, one would expect this optimizer to reach deeper parts of the landscape.

However, the number of initializations that converge to high-cost solutions using Nelder-Mead seems surprisingly high. To examine whether the final parameter values after the initial Nelder-Mead optimization were actual local minima, the worst-performing optimizations' final parameter values were used as the initial parameters in a small simulation using Adam on an ideal simulator. The simulation yielded a lower cost, showing that sample noise hinders the gradient-free optimizers from completely converging to a local minimum. Similar issues in convergence are also present with the SPSA optimizer, as shown by the band of costs ranging between $[-0.7, -0.6]$ in figure 34. It must be noted that with stochastic fluctuations in the cost function, the convergence properties of general SPSA methods are only guaranteed if the starting point of the optimization is in the domain of attraction of the problem [62]. The bulk of the initializations might get stuck in regions of the cost function where convergence is difficult.

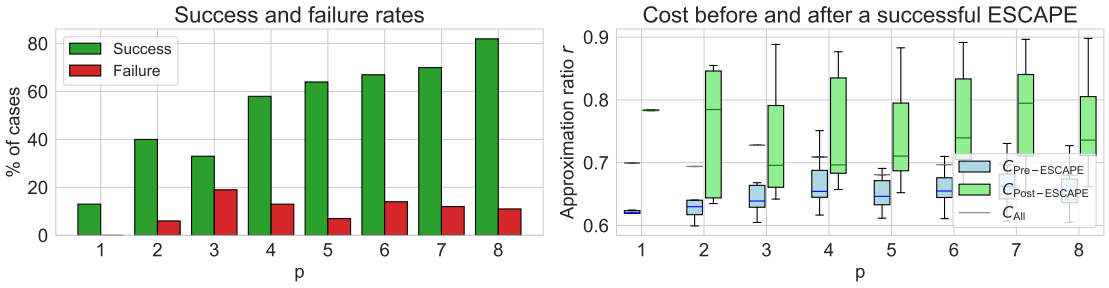
The lack of convergence in these methods is difficult to pinpoint; however, there is reason to believe that the noise levels stemming from shot noise can cause the issue. Firstly, it is important to point out that function comparisons are the basis for taking steps in gradient-free methods; hence a significant amount of noise causes inaccurate comparisons. This issue is most prominent in regions where the difference in cost cannot be differentiated from noise. Examples are flat regions of the cost landscape or around local minima, where the gradient becomes increasingly flatter. As elaborated on further in appendix C, the number of shots needed to get an accuracy of ε scales like $\mathcal{O}(\varepsilon^{-2})$. As mentioned in the implementation part of this section, the number of shots used to estimate the cost function was set to 10 000; hence the errors in the cost function would correspondingly be around 0.01. This was confirmed numerically by calculating the standard fluctuation of a cost value evaluation. The fluctuation for a single $\langle C(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$ was calculated using 50 function evaluations of the same parameter. This was repeated for 100 different $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ configurations to calculate the average standard deviation for these function evaluations. These tests found that the average standard deviation fluctuated at the third decimal for each of the 100 simulations and that the mean standard fluctuation over all 50 tests was 0.0112.

Based on these fluctuations, the same success criteria of 0.1 is used to categorize a successful escape as before. The justification for this choice is that a difference in cost value of 0.1 would categorize a point with a cost value of order ten larger than the shot noise, constituting a point deeper in the landscape.

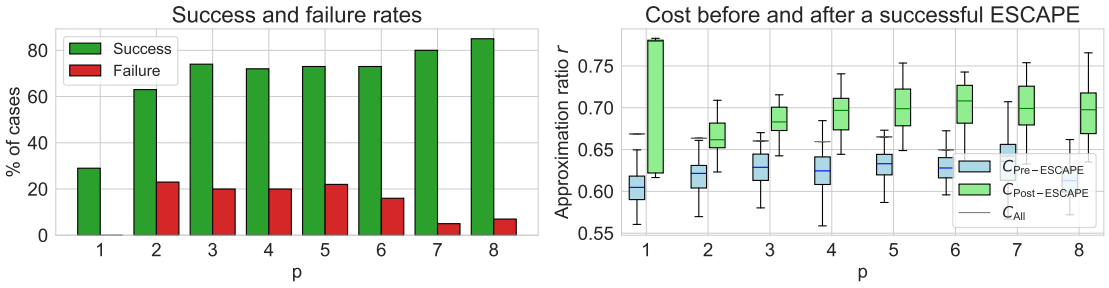
Using this metric to measure a successful escape, figure 35 shows the performance of the gradient-free versions of ESCAPE compared to the original version of the



(a) Results using gradient-based Adam optimizer through the entire procedure



(b) Results using gradient-free SPSA optimizer through the entire procedure

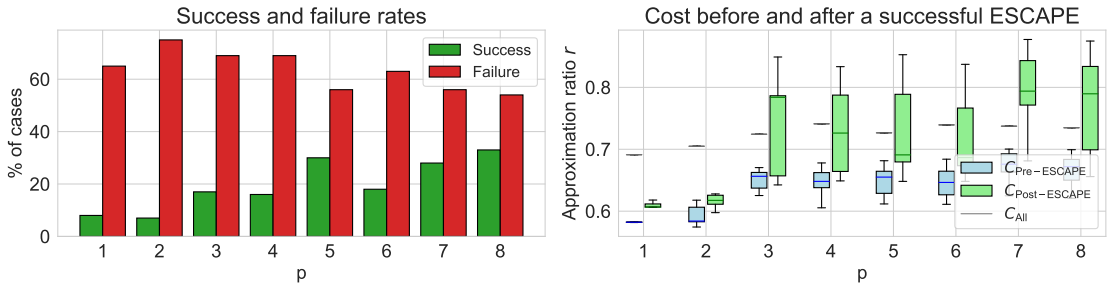


(c) Results using gradient-free Nelder-Mead optimizer through the entire procedure

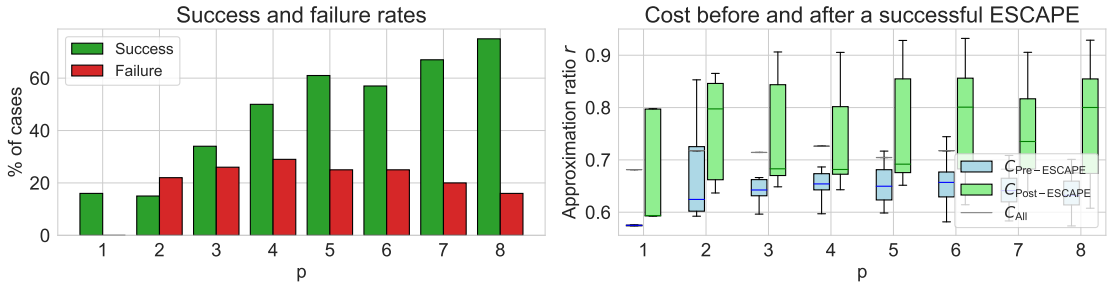
Figure 35: The performance of the gradient-free version of ESCAPE (figures 35b, 35c) compared to the gradient-based ESCAPE procedure using Adam (figure 35a). A maximum of five steps were taken in the gradient-free procedures during the relaxation which lasted for $T = 800$ steps. The neural network was trained for $M = 200$ steps.

algorithm for 100 different random initializations. As the figure shows, the number of successful escapes using the gradient-free procedure is significantly higher than the original ESCAPE method using an ideal simulator and the Adam optimizer. In contrast to ESCAPE on an ideal simulator, the gradient-free versions of the procedure produce a better cost solution a majority of the time for $p > 4$ with success rates as high as 80% for the deepest QAOA depth tested.

Based on the high number of successful escapes from figure 35c, one would naively think that the ESCAPE procedure is most effective when using Nelder-Mead as the gradient-free optimizer. However, the newly found local minimum post-ESCAPE is not significantly deeper than the cost value of the local minimum pre-ESCAPE, indicating that Nelder-Mead quickly converges to sub-optimal local minima. This is most likely because simplex methods are more prone to getting stuck in a local minimum once they are encountered. Adam can escape some of the local minima it



(a) Results using gradient-free SPSA optimizer with $a = 1.5, c = 0.15$



(b) Results using gradient-free SPSA optimizer with $a = 0.2, c = 0.15$

Figure 36: The SPSA version of ESCAPE applied to graph instance E, highlighting the importance of choosing good parameter values a, c used in the SPSA optimizer. In these simulations, $g(t) = \frac{t}{T}$, $T = 800$, $M = 200$ and three SPSA optimization steps are taken during each t.

encounters through momentum and adaptive step sizes, while the random perturbation in SPSA may find a random direction that results in lower cost.

Additionally, based on the convergence issues discussed earlier, one might also argue that the number of successful escapes is inflated as there is some uncertainty as to whether the optimization has fully converged. As shown using graph A in the previous section, this can significantly affect the performance statistics of the ESCAPE routine.

With these limitations in mind, the SPSA version of the procedure seems significantly more promising than the Nelder-Mead version. In terms of the histogram plots, the performance of the procedure is similar to the Nelder-Mead version. However, the SPSA version is capable of performing deeper jumps, as evident by the higher values of the mean and whiskers of the $C_{\text{Post-Escape}}$ in figure 35b. These improvements in cost are comparable to the gradient-based ESCAPE using the Adam optimizer. Therefore, approximating the entire gradient using two function evaluations seems effective in keeping the procedure within the attractive basin during the changing landscape.

In contrast to the gradient-based version of ESCAPE, this shot-based version of the procedure is significantly more sensitive to the hyperparameters of the optimizer; hence this has to be carefully chosen. Recall from section 2.4.2 that there are effectively two parameters that can be altered when using the SPSA optimizer; a essentially acts as the learning rate while c is the scaling of the random shift. Using Spall's [27] parameter initialization as guideline and testing out various configura-

tions of the a and c parameters, it is found that $a = 0.2$ and $c = 0.15$ yield smooth optimizations. To illustrate the importance of choosing correct parameters, figure 36 shows the performance of ESCAPE when a is altered from 0.2 to 1.5. As shown, taking three larger steps during step 4 of the procedure produces worse minima after ESCAPE. This can be attributed to the observation that the optimization is perturbed out of the attractive basin produced by the neural network due to the larger steps taken at step 4 of the procedure.

7.7 ESCAPE with a Noise Model

This section considers a simple simulation with a noise model to see the effects of quantum noise. Pennylane allows for the modeling of noise through the Kraus-matrices formalism. The custom noise model considers two single-qubit quantum error terms, namely depolarizing channels and bit-flip operations. After each QAOA layer, depolarizing channel noise with probability λ is applied. An actual quantum device would have errors at each operation, which is effectively modeled by destroying information after each QAOA layer by depolarizing the qubit. After the last layer, bit-flip noise with equal probability λ is added. This last layer of bit-flip represents measurement errors associated with performing measurements of a quantum system. Formally, the depolarizing noise can be represented by

$$\Delta_\lambda(\rho) = \sum_{i=0}^{3} K_i \rho K_i^\dagger \tag{126}$$

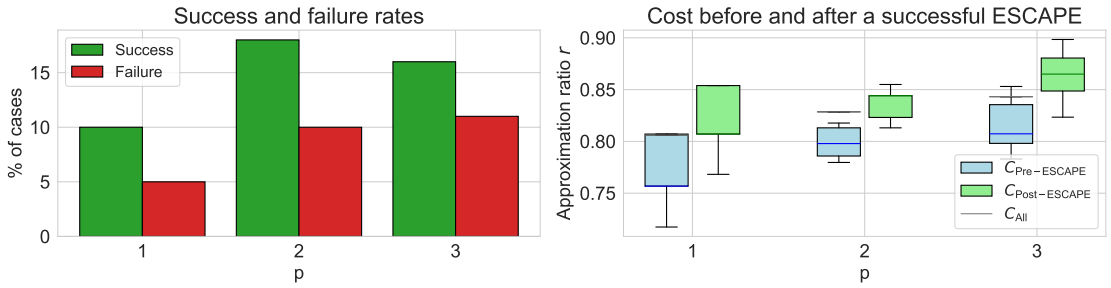
$$K_0 = \sqrt{1 - \frac{3\lambda}{4}} I, K_1 = \sqrt{\frac{\lambda}{4}} X, K_2 = \sqrt{\frac{\lambda}{4}} Y, K_3 = \sqrt{\frac{\lambda}{4}} Z \tag{127}$$

In essence, the Pauli matrices are acted on the input state with probability $\lambda/4$. The bitflip operation is given by the following Kraus matrices:

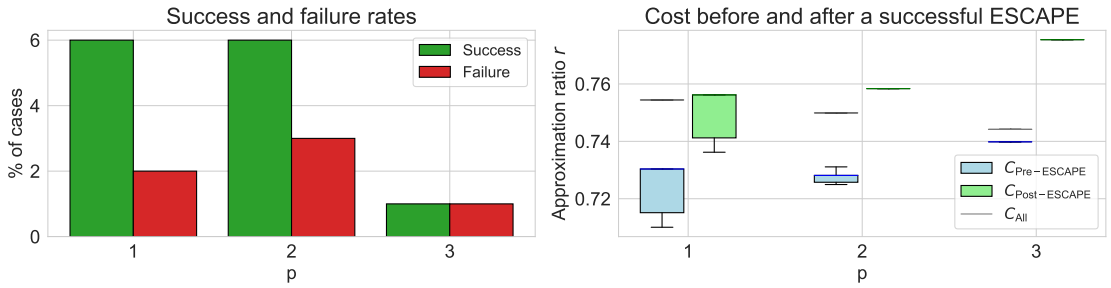
$$K_0 = \sqrt{1-p} I, \quad K_1 = \sqrt{p} X. \tag{128}$$

Two sets of simulations were used to test the ESCAPE procedure on quantum noise on an ideal simulator. In these simulations, the parameter-shift rule was used as the gradient function for the parameters and the Adam optimizer was used during the procedure. Due to computational limitations, only simulations up to QAOA depth $p = 3$ were performed on a five-node w3R graph with weights sampled from $[0, 1]$.

Even at low depths, there are some findings that are worth pointing out. On the ideal simulators, it seems that the processes are able to converge to local minima when the noise levels λ are low, even at higher QAOA depths as shown in figure 37a. This makes sense considering that the depolarizing noise is low at each depth, hence each noise operation primarily consists of applying the identity transformation. As the noise level is increased, each depolarization operation causes the qubits to become



(a) ESCAPE performance when using $g(t) = \frac{t}{T}$ and noise level $\lambda = 0.01$



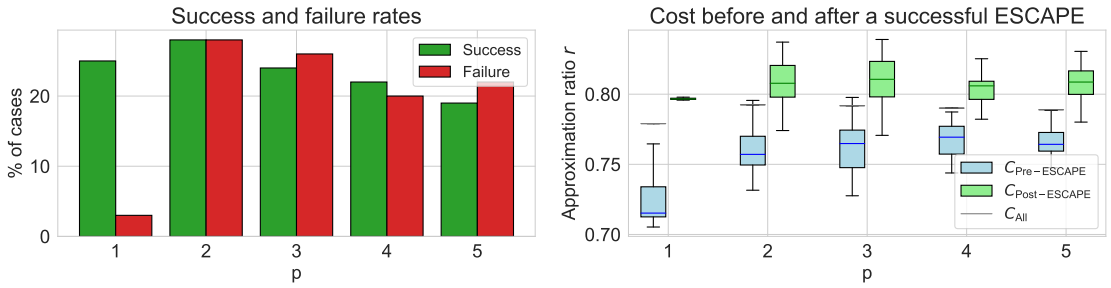
(b) ESCAPE performance when using $g(t) = \frac{t}{T}$ and noise level $\lambda = 0.1$

Figure 37: Comparison between simulations with noise levels of $\lambda = 0.01$ and $\lambda = 0.1$ using the noise model as defined in the text. SGD was used to train the neural network with a learning rate of 0.05 while Adam was used for the variational parameters of the quantum computer.

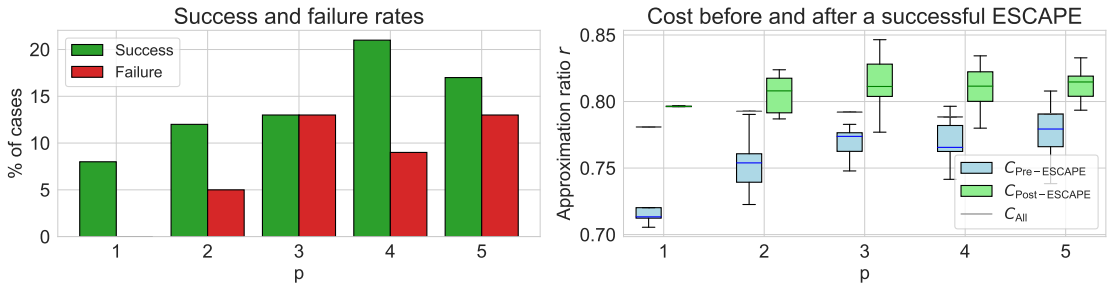
closer to the mixed state, and subsequently, the correct solutions to the problem cannot be amplified properly using the QAOA routine. This is seen from figure 37b where the cost values found are worse as the depth of the circuit increases as shown by C_{All}. When such levels of noise are present in the simulations, the probability of escaping a local minimum is particularly low as indicated by figures 37b. The low jump rate is most likely associated with the gradient being inaccurate, and hence the ESCAPE causes random walks in the close vicinity of the originally found local minimum. Additionally, the noise would cause the landscape to inherently become different which may also cause the procedure to fail.

With regard to noise it is more interesting to see how the procedure fares on real quantum hardware. This was investigated using the 5-qubit FakeManilla noise model from IBMQ. The setup is the same as the shot-based procedure used throughout the thesis. Due to a large amount of noise and uncertainty in hardware-based simulators, the SPSA algorithm was used to perform the initial and final optimization. Two simulations were run with this particular noise model, one using $g(t) = \frac{t}{T}$ and another using the Heaviside function $g(t) = \Theta(t-150)$. When using $g(t) = \Theta(t-150)$ the SPSA optimization was performed using 150 steps. For the t/T relaxation, T was set to 350 and 10 SPSA steps were taken at each t to approximate a single gradient step.

Consider first the noise level present in the FakeManilla noise model compared to the custom depolarizing noise model. It seems that the level of noise inherent in the hardware may be simulated using an error-probability λ that is somewhere



(a) ESCAPE performance when using $g(t) = \frac{t}{T}$



(b) ESCAPE performance when using $g(t) = \Theta(t - T)$

Figure 38: Comparison between $g(t) = \frac{t}{T}$ versus $g(t) = \Theta(t - T)$ on the IBM fakemanilla noisemodel. SGD was used to train the neural network with a learning rate of 0.05 while SPSA was used for the variational parameters of the quantum computer. T was set to 350 for $g(t) = \frac{t}{T}$ and 150 for $g(t) = \Theta(t - T)$.

between $[0.01, 0.1]$. This is evident from the observation that the achieved approximation ratios from the FakeManilla simulations lie between $[0.7, 0.85]$. This is to be compared to the approximation ratios ranging between $([0.75, 0.9], [0.72, 0.76])$ for $\lambda = (0.01, 0.1)$ respectively.

As figure 38 shows, the performance of the ESCAPE routine for both cases is mixed. Firstly, a familiar trend reappears when comparing these simulations to the ones with graph instance A. Namely, both success and deterioration rates are generally higher when training in the dynamic landscape using $g(t) = t/T$. However, in contrast to the previously seen trends in graph instance A, the deterioration rates are comparable to the number of successful escapes. The reason for the higher deterioration rates with the dynamically changing landscape stems from the parameter vector's inability to remain in the local minimum as the landscape changes. Due to both shot noise and quantum noise from the device, the procedure is more prone to falling out of this local minimum, giving rise to high numbers of unsuccessful escapes.

As the results indicate the ESCAPE routine is mostly unsuccessful in finding better local minima when noise is present in the circuit. Most of the initializations tend to find the same local minimum that was initially found. Given that a new minimum is found, the probability of it being an improvement is comparable to it being worse. Additionally, given that a better minimum is found, the improvement in cost is not that significant. It is difficult to draw anything conclusive from these results as the graph instances are still relatively small. However, it seems that lower levels of noise are needed to get any significant improvements when applying the ESCAPE

procedure on NISQ hardware.

8 Conclusion and outlook

This thesis presented a high-level overview of machine learning, quantum machine learning, and issues concerning the trainability of quantum circuits. Using two different heuristics for parameter initialization, INTERP and Parameter-fixing, the variational quantum algorithm QAOA was applied to the classical binary optimization problem MaxCut for various graph instances. Results from numerical simulations show that both initialization heuristics consistently outperform randomized initializations of the QAOA parameters. Additionally, comparisons showed that both yield the same expected behavior when the heuristics succeed. For both, it is found that the gate parameters tend to remain relatively fixed as the depth increases. Some of the heuristics' success is attributed to the emergence of minimum cost valleys, making optimization easier.

Based on the findings from the above parameter initialization heuristics, a feedforward neural network was used to form predictions of the optimal QAOA patterns from 200 different twelve-node w3R graph instances. Correlation between the neural network features and the output shows that there are limitations to the predictive power of only using two features to form predictions of all QAOA parameters at the final depth. This limitation was also evident from the relative error between the target variable and the neural network's predictions. This was particularly the case for the β parameters, where the relative errors for some predictions reached levels as high as 100%. Despite the significant errors, using these parameters as the initial point for a final optimization would find the exact optimum in the dataset, hence validating this approach. This way of utilizing a neural network has the advantage of skipping intermediate optimization steps used in heuristics such as INTERP and parameter-fixing, provided that a dataset is available to train on. Potential extensions to this method are to consider deeper networks, find other features that show strong correlations with the target variable, or use some other different machine learning framework to form predictions.

The next part of the thesis considered the ESCAPE algorithm, where a neural network was used to escape potential local minima in the cost landscape once encountered. Results showed a lack of performance when the procedure was applied to smaller graph instances. The best performing ESCAPE runs involved a not overly trained neural network, with its contribution gradually turned off using $g(t) = t/T$. The pattern of increasing success ratio found by Riveradean et al. [4] was not as general for the minor graph instances unless the initial optimization was terminated early, or the graph had both positive and negative graph partitions. For these smaller graph instances, the ESCAPE procedure was ineffective as most runs found the same local minimum it started in.

The trend of increasing successful escapes with QAOA depth does seem to hold as the graph size increases. This was tested using the ideal simulator and the shot-based procedure that used gradient-free steps instead of a gradient-based one. It was found that the gradient-free version of ESCAPE on a shot-based simulator significantly increased the number of successful escapes. However, there is reason to be cautious about the increased performance. As elaborated in section 7.6, it

is challenging to separate convergence to a local minimum from regions where the cost decreases slowly. The SPSA minimizer, in particular, ran into this problem where the cost would decrease slowly throughout the initial optimization. With these limitations in mind, improvements in cost using the ESCAPE procedure with the SPSA optimizer are quite significant and comparable to the jumps performed by the Adam optimizer on an ideal simulator. Lastly, the ESCAPE procedure was also run using realistic hardware noise. In these cases, it was found that the number of failed ESCAPE attempts was comparable to the successful ones; hence the method is less reliable on NISQ hardware.

The thesis has demonstrated that finding reasonable solutions to the MaxCut problem can be difficult, even at intermediate depths. This is due to the cost landscape having several non-optimal local minima, which the optimization tends to get stuck in. The ESCAPE procedure is meant to aid in this process; however, the ideal simulator's findings seem to suggest that a significant amount of initializations will not find an improvement in cost. Moreover, the gradient-free procedure performance has inflated numbers due to a lack of convergence during the initial optimization. Still, there is no guarantee that the procedure will significantly improve the cost. Therefore, this thesis further reiterates that using a randomized initialization strategy to solve VQA problems is ineffective, even when methods like ESCAPE are used to correct some of them.

Therefore, if one wishes to find suitable parameters to the variational QAOA ansatz that generate bitstrings that solve the MaxCut problem with high probability, one should use a systematic approach. An example of such an approach is Zhou et al.'s [1] INTERP heuristic, which works for w3R, u3R, and Erdös Renyi graph instances, as tested in the thesis. It is important to note that this is merely a heuristic and that one must conduct similar studies as Zhou et al. [1] to gauge the performance of other graph types. If such patterns generally hold, one can readily use classical machine learning to learn the trends in the optimal QAOA parameters to reduce necessary quantum resources. Out of the methods discussed throughout the thesis, the machine learning approach using INTERP as the basis for the generated dataset seems the most effective to solve the MaxCut graph problem using the QAOA ansatz. Similar approaches using machine learning in this capacity seem like a promising direction for future work.

A Appendix: Derivation of generalization error

The derivation uses methods from statistical learning and follows the one presented by Weinberger [63]. It will be shown that there is a trade-off between a model being flexible enough to fit the training data, but also that given too much model complexity, the generalization error will start to increase.

As with any supervised learning task, a set of data $\mathcal{D} = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^m, y^m)\}$ is drawn iid from some underlying probability distribution $P(X, Y)$, and assume that the problem is a regression problem. Using this dataset as a training set, a machine learning algorithm $\mathcal{A}$ (ex. neural network, linear regression, etc.) is used on this dataset to fit a model $f_D = \mathcal{A}(D)$. Additionally, there is not necessarily a unique label y for each feature vector $\boldsymbol{x}$, thus two identical feature vectors can give the same label y. An example of such a case can be two houses with identical features sold at different prices. Some notation is needed to show the decomposed generalization error.

Expected label (given $\boldsymbol{x} \in \mathbb{R}^d$):

$$\bar{y}(\boldsymbol{x}) = E_{y|\boldsymbol{x}}[Y] = \int_y y \, \mathrm{P}(y|\boldsymbol{x}) dy \tag{129}$$

This captures the notion that one feature vector $\boldsymbol{x}$ may give rise to different labels y, hence the expected output *given* an input $\boldsymbol{x}$ is given by $\bar{y}(x)$, hence the $P(y|x)$ in the above expression..

Expected Test error (given f_D):

$$E_{(\boldsymbol{x},y)\sim P}\left[(f_D(\boldsymbol{x}) - y)^2\right] = \iint_{xy} (f_D(\boldsymbol{x}) - y)^2 \, \mathrm{P}(\boldsymbol{x}, y) dy d\boldsymbol{x} \tag{130}$$

Given that a model is fit from the model family $\mathcal{A}$, the generalization error of the fitted model can be computed from the above expression (hence the given f_D caution). This expression captures the generalization of the fitted model because new points $(\boldsymbol{x}, y)$ are sampled from the true distribution to compute the expected loss, hence a test-set is being used to calculate the loss over new, unseen instances. Note here that the squared loss has been used as a measure of error.

This expression is valid for a fixed data set D from which the model function f_D was created. However, a differently sampled data set would generate different models f_D. Hence, it is meaningful to consider the average model generated over several independently sampled data sets.

Expected model (given $\mathcal{A}$):

$$\bar{f} = E_{D\sim P^n}\left[f_D\right] = \int_D f_D \, \mathrm{P}(D) dD \tag{131}$$

Here, $\mathrm{P}(D)$ refers to the probability of drawing a particular data set by sampling n pairs $(\boldsymbol{x}, y)$ from the original probability distribution $P(X, Y)$. Now, taking the

expectation value over the expected test error given a certain model f_D and integrating over all possible combinations of sampled data sets D gives the expected test error given a particular choice of the model family $\mathcal{A}$.

Expected Test error (given $\mathcal{A}$):

$$\underset{\substack{(\boldsymbol{x},y)\sim P \\ D\sim P^n}}{E}\left[(f_D(\boldsymbol{x})-y)^2\right] = \int_D\int_{\boldsymbol{x}}\int_y (f_D(\boldsymbol{x})-y)^2\,\mathrm{P}(\boldsymbol{x},y)\mathrm{P}(D)d\boldsymbol{x}dydD \tag{132}$$

Note here that the variables $(\boldsymbol{x}, y)$ represent the test set as they are independently sampled from the underlying distribution, while the training set D is used to create the model. The set of points constituting the training set was used to create the model f_D, and are hence different points from the variables $(\boldsymbol{x}, y)$.

This is the expression that represents the generalization error as it evaluates the test error of the model family $\mathcal{A}$ with respect to the underlying data distribution $P(X, Y)$. By expanding the square and evaluating the individual terms it can be shown that the generalization error can be decomposed into three interpretable terms.

$$\begin{aligned}
E_{\boldsymbol{x},y,D}\left[[f_D(\boldsymbol{x})-y]^2\right] &= E_{\boldsymbol{x},y,D}\left[\left[\left(f_D(\boldsymbol{x})-\bar{f}(\boldsymbol{x})\right)+\left(\bar{f}(\boldsymbol{x})-y\right)\right]^2\right] \\
&= E_{\boldsymbol{x},D}\left[\left(\bar{f}_D(\boldsymbol{x})-\bar{f}(\boldsymbol{x})\right)^2\right] \\
&+ \underbrace{2E_{\boldsymbol{x},y,D}\left[\left(f_D(\boldsymbol{x})-\bar{f}(\boldsymbol{x})\right)\left(\bar{f}(\boldsymbol{x})-y\right)\right]}_{0} + E_{\boldsymbol{x},y}\left[\left(\bar{f}(\boldsymbol{x})-y\right)^2\right]
\end{aligned}$$

The second term is zero since the $E_D[f_D(x) - \bar{f}(x)] = 0$. Now, evaluate the last term by expanding the square adding and subtracting $\bar{y}(\boldsymbol{x})$:

$$\begin{aligned}
E_{\boldsymbol{x},y}\left[(\bar{f}(\boldsymbol{x})-y)^2\right] &= E_{\boldsymbol{x},y}\left[(\bar{f}(\boldsymbol{x})-\bar{y}(\boldsymbol{x}))+(\bar{y}(\boldsymbol{x})-y)^2\right] \\
&= E_{\boldsymbol{x},y}\left[(\bar{y}(\boldsymbol{x})-y)^2\right] + E_{\boldsymbol{x}}\left[(\bar{f}(\boldsymbol{x})-\bar{y}(\boldsymbol{x}))^2\right] \\
&+ \underbrace{2E_{\boldsymbol{x},y}[(\bar{f}(\boldsymbol{x})-\bar{y}(\boldsymbol{x}))(\bar{y}(\boldsymbol{x})-y)]}_{0}
\end{aligned}$$

Putting all this together give three different terms for the generalization error as elaborated on further in the main text.

$$E_{\boldsymbol{x},y,D}\left[(f_D(\boldsymbol{x})-y)^2\right] = \underbrace{E_{\boldsymbol{x},D}\left[\left(f_D(\boldsymbol{x})-\bar{f}(\boldsymbol{x})\right)^2\right]}_{\text{Variance}} \tag{133}$$

$$+ \underbrace{E_{\boldsymbol{x},y}\left[(\bar{y}(\boldsymbol{x})-y)^2\right]}_{\text{Noise}} + \underbrace{E_{\boldsymbol{x}}\left[(\bar{f}(\boldsymbol{x})-\bar{y}(\boldsymbol{x}))^2\right]}_{\text{Bias}^2} \tag{134}$$

B Appendix: Simultaneous measurement procedure

The naive method of measurement is one where each Pauli string is measured separately. For instance, if the four qubit Pauli string $XYZZ$ needed to be measured, one would apply H and $HS^\dagger$ operations on the first and second wires before, then perform measurements in the Z-basis on all wires.

Qubit-wise-commuting (QWC) Pauli strings have a similar measurement procedure. This type of commuting is one in which at each index, the corresponding two Pauli-matrices commute. An example of this is the set of Pauli gates $\{XX, IX, XI, II\}$ since there are only two possibilities, X, I at each index, and $[X, I] = [X, X] = [I, I] = 0$. Measuring these strings reduces to changing the basis in which the qubit-index does commute. For instance, if one were to simultaneously measure the two strings $\{XIYIZI, IXIYIZ\}$, one would simply use the X basis for the two first qubits, Y basis for the middle two qubits, and Z basis for the last two qubits and perform measurements.

The measurement circuit for general commutative (GC) Pauli strings is not trivial as in the other cases. Two Pauli strings are GC if and only if they do not commute on an even number of indices. For example the partition $\{XX, YY, ZZ\}$ is a GC partition since the Pauli matrix at each index is either X, Y, Z, and $[X, Y], [X, Z], [Y, Z] \neq 0$. The statement can be shown as follows. Consider two N-qubit Pauli strings A, B and the product AB,

$$A = \bigotimes_{j=1}^{N} A_j, \quad B = \bigotimes_{j=1}^{N} B_j, \quad A_j, B_j \in \{I, X, Y, Z\} \tag{135}$$

$$AB = \bigotimes_{j=1}^{N} A_j B_j = \bigotimes_{j=1}^{N} \left\{ \begin{array}{ll} B_j A_j & \text{if } [A_j, B_j] = 0 \\ -B_j A_j & \text{if } [A_j, B_j] \neq 0 \end{array} \right\} = (-1)^k BA \tag{136}$$

where k is the number of indices where the Pauli-matrices do not commute. Here it was used that $A_i B_i = -B_i A_i$ for non-commuting matrices due to the anti-commutation relation between Pauli matrices. For the two operators to commute, AB must be equal to BA which implies that k is even.

As mentioned, performing simultaneous measurements on GC Pauli-strings is non-trivial. This cannot be performed naively by rotating the axes of the shared eigenvectors to align with the standard Z-axis since the column vector is of size 2^N. However, by exploiting the symmetries of Pauli-matrices, the transpilation algorithm of Gokhale et al. [31] is able to synthesize a circuit which can perform simultaneous measurements. The crucial insight behind this procedure is *unitary conjugation*, namely that after applying a quantum gate U, a target measurement on the original state becomes equivalent to the measurement of UMU^{-1} by the following,

$$\langle\psi|M|\psi\rangle = \langle\psi|U^{-1}UMU^{-1}U|\psi\rangle = \langle\tilde{\psi}|UMU^{-1}|\tilde{\psi}\rangle \tag{137}$$

where it was used that quantum gates U are unitary and $|\tilde{\psi}\rangle = U|\psi\rangle$ is the transformed state. The essential goal of the mapping is to create a unitary which is able to transform a commuting family of Pauli operators into the computational basis, given by $\{ZI\cdots I, IZ\cdots I, \ldots, I\cdots ZI, I\cdots IZ\}$. In other words, what this procedure entails is that by applying this unitary, measuring the *first qubit in the z-basis* measures the outcome of the *first Pauli string* in the family while *measuring the second qubit in the z-basis* results in the measurement of the *second Pauli string*, etc. Therefore, one single shot/run on quantum hardware is able to simultaneously measure the results of N commuting Pauli strings through the measurement of each qubit. Before exemplifying this procedure, one has to be familiar with some of the formalism behind "stabilizer matrices".

The stabilizer matrix is a simplified way of representing a set of Pauli strings. Describing the formalism is easier through an example. Given a set of Pauli operators, $[XXX, YYY, ZZZ, XYZ]$, the stabilizer matrix is given by

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \tag{138}$$

The dimension of the stabilizer matrix is given by $2N \times P$ where N is the amount of pauli-terms within a string (3 in this example) and P is the number of Pauli strings in the family (4 in this example). The first N rows are referred to as the Z_S-matrix while the lower N rows are referred to as the X_S-matrix. Each of the members in the family are represented by the column vectors of the stabilizer matrix. For instance, the first column represents the XXX operator, second column represents YYY operator, etc. The i,j-th element of the Z_S matrix are given by whether the index i of the j-th pauli string in the family is a pauli Z operator. Correspondingly for the X_S matrix where the string is considered to be a pauli X matrix instead. If there is a Y operator, there is a 1 in both X_S and Z_S because of the pauli relation $Y = -iZX$. Considering the last 3-qubit operator in this example, XYZ, the column vector in the stabilizer matrix is given by $[0,1,1,1,1,0]$. The column vector representing The Z_S, X_S matrices therefore have column vectors $[0,1,1]$ and $[1,1,0]$ respectively. This indicates that there is an X operator on index 1 (0 in Z_S, 1 in X_S), Y operator on index 2 (1 in both matrices) and Z operator on index 3 (1 in Z_S, 0 in X_S). Similar argumentation gives the remaining column vectors of the stabilizer matrix.

As mentioned in the main text, the goal of the algorithm is to transform an arbitrary set of commuting Pauli family into the computational basis. In other word, they are

transformed into $\{ZI\cdots I, IZ\cdots I, \ldots, I\cdots ZI, I\cdots IZ\}$. The approach presented by Gokhale et al. is therefore to transform a given stabilizer matrix into a stabilizer matrix where the first N rows are given by $N \times N$-dimensional identity and the elements of the lower matrix are all zeros. Note that the element of min-clique cover comes into play in this sense because the Z_S and X_S are going to be square matrix since the remaining matrices in the family can be given by products of some of the matrices within the family as shown in the example of the main text.

There are five different logic gates needed to perform the transformation, namely the single qubit gates H, S and two-qubit gates $CZ, \mathrm{SWAP}, \mathrm{CNOT}$. The effect of these gates can be calculated by explicit matrix multiplication.

	$UZU^\dagger$	$UXU^\dagger$
$U = H$	X	Z
$U = S$	Z	Y

(139)

	$UZIU^\dagger$	$UIZU^\dagger$	$UXIU^\dagger$	$UIXU^\dagger$
$U = CNOT$	ZI	ZZ	XX	IX
$U = CZ$	ZI	IZ	XZ	ZX
$U = SWAP$	ZI	ZI	IX	XI

(140)

The effect of these gates on the stabilizer matrix can be summed up as follows:

- H applied on the i-th qubit swaps i-th and $i + N$-th row of the stabilizer matrix. This essentially swaps the i-th rows of the Z_S and X_S matrices.
- S on the ith qubit sets the (i, i) diagonal entry of the Z_S matrix to 0
- CNOT controlled on the i-th qubit and targeted on the j-th qubit adds j-th row on the i-th row and adds $i + N$-th row to the $j + N$-th row. All additions are performed modulo 2.
- CZ between i-th and j-th qubits sets the (i, j) and (j, i) symmetric off-diagonal elements in the Z_S matrix to 0.
- SWAP between i-th and j-th qubits swap the i-th and j-th row of both X_S and Z_S matrices.

These are the needed ingredients of the stabilizer formalism. Using these gates one can transform a given stabilizer matrix into the stabilizer matrix corresponding to a computational basis measurement.

To exemplify this procedure, let us consider the measurement of the general commuting set of gates, $[XX, YY, ZZ]$. Note first that it is possible to represent $YY = -(XX)(ZZ)$, therefore we can restrict to only measure in the $[XX, ZZ]$ bases. The goal of the algorithm is to create a unitary transformation that maps $[XX, ZZ] \to [ZI, IZ]$ so that a measurement of the first qubit in the computational

basis is equivalent to measuring XX and the second to ZZ. Using stabilizer matrix formalism, the transformation is given by the following:

$$[XX, ZZ] = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \xrightarrow{\text{CNOT}} \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} = [XI, IZ] \tag{141}$$

$$[XI, IZ] = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \xrightarrow{H \otimes I} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} = [ZI, IZ] \tag{142}$$

With this procedure in hand, the only remaining piece is to perform the partition of the Pauli strings of the Hamiltonian into commuting families. This problem can be mapped to the problem of finding the minimum clique cover of a graph [64] where each node represents one of the Pauli strings of the Hamiltonian. The min-clique problem is widely studied in computational complexity theory. The problem is shown to be NP hard, however for small graph instances the problem should be solvable in reasonable amount of time on a classical computer.

C Appendix: Measurement Accuracy

As the simultaneous measurement procedure boils down to measuring single qubits in the computational basis. The expectation value of $\langle Z \rangle$ is performed by running the quantum circuit S times, conventionally called shots, and sampling the corresponding result $\{-1, 1\}$. Multiple runs will yield an estimate of the mean value, and increased shots yield higher accuracy. Since the outcome is either -1 or 1, this sampling procedure can be viewed as sampling from a Bernoulli distribution with probability p, where the probability p is dependent on the quantum state prior to measurement. This probability can analytically calculated by tracing out all other qubits. The error is a deviation from this p, and can be estimated through confidence intervals $[p - \epsilon, p + \epsilon]$. In essence, this problem boils down to a Bernoulli trial; a coin-flip trial with a biased probability towards one of the outcomes. There are several ways of estimating the error in the confidence interval of Bernoulli trials. One of which being the Wilson Score [65] interval. In this interval, the probability estimator and error are given by

$$\hat{\hat{p}} = \frac{1}{1 + \frac{z^2}{S}} \left(\hat{p} + \frac{z^2}{2S} \right) \tag{143}$$

$$\epsilon = \frac{z}{1 + \frac{z^2}{S}} \left(\frac{\hat{p}(1 - \hat{p})}{S} + \frac{z^2}{4S^2} \right)^{\frac{1}{2}}, \tag{144}$$

where $\hat{p}$ is the ratio of samples being in state 1 (empirical probability), z is the confidence level, S is the number of trials (which translates to shots on quantum computer) and ϵ is the deviation from p. The error is maximized at $\hat{p} = 0.5$, giving a bound on the error and an estimation on the number of necessary shots. Inserting this into the expression for the error and solving for the number of shots S yields

$$\epsilon \leq \sqrt{z^2 \frac{S + z^2}{4S^2}} \tag{145}$$

$$S \leq \frac{\epsilon^2 \sqrt{\frac{z^4(16\epsilon^2 + 1)}{\epsilon^4}} + z^2}{8\epsilon^2} \tag{146}$$

which scales like $\mathcal{O}(\epsilon^{-2})$ in the number of shots on the quantum computer to compute a single qubit measurement in the computational basis.

To conclude the section on measurement, Gokhale et al. [31] show that it is possible to reduce the number of needed measurements by separating the problem of measuring the mean value of a Hamiltonian into the sum of measuring multiple Pauli strings. Additionally, rather than needing to measure several Pauli strings one at a time, simultaneous measurement of commuting sets of Pauli strings is possible. Therefore, the general measurement procedure to perform measurements efficiently goes as follows: reduce the Hamiltonian into Pauli strings, group the strings into as

few partitions as possible where each string within a partition form a set of general commuting operators. These groupings are usually made by creating the minimum clique cover over all the Pauli-terms; a widely studied problem in computational complexity theory. Once such groupings have been created, one uses the algorithm proposed by Gokhale et al. [31] to create the needed circuit for each partition. Then, measuring each qubit in the computational basis multiple times yields an estimate of $\langle P_i \rangle$, whose precision scales like $\mathcal{O}(\epsilon^{-2})$. Summing each of these mean values estimates the original Hamiltonian.

References

[1] L. Zhou et al. 'Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices'. In: *Physical Review X* 10.2 (2020) (cit. on pp. i, iii, 3, 51–53, 55, 56, 59, 63, 97).

[2] X. Lee et al. *Parameters Fixing Strategy for Quantum Approximate Optimization Algorithm*. 2021. arXiv: 2108.05288 [quant-ph] (cit. on pp. i, iii, 3, 51–53, 56–58, 60).

[3] M. Alam, A. Ash-Saki and S. Ghosh. 'Accelerating Quantum Approximate Optimization Algorithm using Machine Learning'. In: *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2020, pp. 686–689 (cit. on pp. i, iii, 3, 65).

[4] J. Rivera-Dean et al. *Avoiding local minima in Variational Quantum Algorithms with Neural Networks*. 2021. arXiv: 2104.02955 [quant-ph] (cit. on pp. i, iii, 2, 3, 70, 72, 74, 81, 82, 84–86, 96).

[5] P. W. Shor. 'Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer'. In: *SIAM Journal on Computing* 26.5 (1997), 1484–1509 (cit. on pp. 1, 23).

[6] A. W. Harrow, A. Hassidim and S. Lloyd. 'Quantum Algorithm for Linear Systems of Equations'. In: *Physical Review Letters* 103.15 (2009) (cit. on p. 1).

[7] J. Preskill. 'Quantum Computing in the NISQ era and beyond'. In: *Quantum* 2 (2018), p. 79 (cit. on p. 1).

[8] A. Peruzzo et al. 'A variational eigenvalue solver on a photonic quantum processor'. In: *Nature Communications* 5.1 (2014) (cit. on p. 1).

[9] E. Farhi, J. Goldstone and S. Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028 [quant-ph] (cit. on pp. 1, 47, 50, 55).

[10] L. Bittel and M. Kliesch. 'Training Variational Quantum Algorithms Is NP-Hard'. In: *Physical Review Letters* 127.12 (2021) (cit. on p. 1).

[11] E. R. Anschuetz. *Critical Points in Hamiltonian Agnostic Variational Quantum Algorithms*. 2021. arXiv: 2109.06957 [quant-ph] (cit. on pp. 1, 43, 44).

[12] B. T. Kiani, S. Lloyd and R. Maity. *Learning Unitaries by Gradient Descent*. 2020. arXiv: 2001.11897 [quant-ph] (cit. on pp. 2, 44).

[13] Y. Bahri et al. 'Statistical Mechanics of Deep Learning'. In: *Annual Review of Condensed Matter Physics* 11.1 (2020), pp. 501–528 (cit. on pp. 2, 5, 70).

[14] A. Choromanska et al. *The Loss Surfaces of Multilayer Networks*. 2015. arXiv: 1412.0233 [cs.LG] (cit. on p. 2).

[15] T. M. Mitchell. *Machine learning*. McGraw Hill, 1997 (cit. on p. 4).

[16] M. Schuld and F. Petruccione. *Supervised Learning with Quantum Computers*. 1st. Springer Publishing Company, Incorporated, 2018 (cit. on pp. 4, 7, 8, 18, 21, 22, 27).

[17] F. Petruccione and M. Schuld. *Machine Learning with Quantum Computers*. Springer Nature, 2021 (cit. on p. 5).

[18] S. Dwivedi and L. K. P.Bhaiya. 'A Systematic Review on K-Means Clustering Techniques'. In: 2019 (cit. on p. 6).

[19] F. Murtagh and P. Contreras. *Methods of Hierarchical Clustering*. 2011. arXiv: 1105.0121 [cs.IR] (cit. on p. 6).

[20] T. Rigon, A. H. Herring and D. B. Dunson. *A generalized Bayes framework for probabilistic clustering*. 2020. arXiv: 2006.05451 [stat.ME] (cit. on p. 6).

[21] J. Shlens. *A Tutorial on Principal Component Analysis*. 2014. arXiv: 1404.1100 [cs.LG] (cit. on p. 6).

[22] B. Bermeitinger, T. Hrycej and S. Handschuh. 'Singular Value Decomposition and Neural Networks'. In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning* (2019), 153–164 (cit. on p. 6).

[23] D. Bank, N. Koenigstein and R. Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG] (cit. on p. 6).

[24] J. Wang. *An Intuitive Tutorial to Gaussian Processes Regression*. 2021. arXiv: 2009.10862 [stat.ML] (cit. on p. 6).

[25] M. Belkin et al. *Reconciling modern machine learning practice and the bias-variance trade-off*. 2019. arXiv: 1812.11118 [stat.ML] (cit. on pp. 12, 13).

[26] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (cit. on p. 14).

[27] J. Spall. 'Implementation of the simultaneous perturbation algorithm for stochastic optimization'. In: *IEEE Transactions on Aerospace and Electronic Systems* 34.3 (1998), pp. 817–823 (cit. on pp. 15, 16, 91).

[28] L. K. Grover. 'A fast quantum mechanical algorithm for database search'. In: *arXiv e-prints*, quant-ph/9605043 (May 1996), quant–ph/9605043. arXiv: quant-ph/9605043 [quant-ph] (cit. on p. 23).

[29] D. Camps, R. Van Beeumen and C. Yang. 'Quantum Fourier transform revisited'. In: *Numerical Linear Algebra with Applications* 28.1 (2020) (cit. on p. 23).

[30] M. Schuld. *Supervised quantum machine learning models are kernel methods*. 2021. arXiv: 2101.11020 [quant-ph] (cit. on p. 24).

[31] P. Gokhale et al. 'Optimization of Simultaneous Measurement for Variational Quantum Eigensolver Applications'. In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2020, pp. 379–390 (cit. on pp. 29, 100, 104, 105).

[32] Y. Shikano et al. 'Post-Hartree–Fock method in quantum chemistry for quantum computer'. In: *The European Physical Journal Special Topics* 230.4 (2021), 1037–1051 (cit. on p. 30).

[33] M. A. Nielsen. 'The Fermionic canonical commutation relations and the Jordan-Wigner transform'. In: 2005 (cit. on p. 30).

[34] S. Hadfield et al. 'From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz'. In: *Algorithms* 12.2 (2019), p. 34 (cit. on p. 31).

[35] Z. Holmes et al. *Connecting ansatz expressibility to gradient magnitudes and barren plateaus*. 2021. arXiv: 2101.02138 [quant-ph] (cit. on pp. 31, 32, 41).

[36] S. Sim, P. D. Johnson and A. Aspuru-Guzik. 'Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms'. In: *Advanced Quantum Technologies* 2.12 (2019), p. 1900070 (cit. on p. 33).

[37] Pennylane. *Unitary Designs*. 2021 (cit. on p. 33).

[38] M. Gerken. 'Measure concentration : Levy ' s Lemma Lecture notes for Talk 6'. In: 2013 (cit. on p. 34).

[39] P. Hayden, D. W. Leung and A. Winter. 'Aspects of Generic Entanglement'. In: *Communications in Mathematical Physics* 265.1 (2006), 95–117 (cit. on p. 34).

[40] M. Schuld et al. 'Evaluating analytic gradients on quantum hardware'. In: *Physical Review A* 99.3 (2019) (cit. on p. 35).

[41] A. G. Baydin et al. *Automatic differentiation in machine learning: a survey*. 2018. arXiv: 1502.05767 [cs.SC] (cit. on p. 35).

[42] A. Mari, T. R. Bromley and N. Killoran. 'Estimating the gradient and higher-order derivatives on quantum hardware'. In: *Physical Review A* 103.1 (2021) (cit. on p. 36).

[43] V. Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2020. arXiv: 1811.04968 [quant-ph] (cit. on pp. 38, 53).

[44] L. Banchi and G. E. Crooks. 'Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule'. In: *Quantum* 5 (2021), p. 386 (cit. on p. 38).

[45] D. Wierichs et al. *General parameter-shift rules for quantum gradients*. 2021. arXiv: 2107.12390 [quant-ph] (cit. on pp. 39, 40).

[46] M. Cerezo et al. 'Cost function dependent barren plateaus in shallow parametrized quantum circuits'. In: *Nature Communications* 12.1 (2021) (cit. on p. 42).

[47] S. Wang et al. *Noise-Induced Barren Plateaus in Variational Quantum Algorithms*. 2021. arXiv: 2007.14384 [quant-ph] (cit. on p. 42).

[48] R. Wiersema et al. 'Exploring Entanglement and Optimization within the Hamiltonian Variational Ansatz'. In: *PRX Quantum* 1.2 (2020) (cit. on pp. 42, 44).

[49] A. J. Bray and D. S. Dean. 'Statistics of Critical Points of Gaussian Fields on Large-Dimensional Spaces'. In: *Physical Review Letters* 98.15 (2007) (cit. on p. 43).

[50] Y. Dauphin et al. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*. 2014. arXiv: 1406.2572 [cs.LG] (cit. on p. 43).

[51] E. Farhi and A. W. Harrow. *Quantum Supremacy through the Quantum Approximate Optimization Algorithm*. 2019. arXiv: 1602.07674 [quant-ph] (cit. on pp. 45, 47).

[52] J. Håstad. 'Some Optimal Inapproximability Results'. In: *J. ACM* 48.4 (July 2001), 798–859 (cit. on p. 47).

[53] S. Arora et al. 'Proof Verification and the Hardness of Approximation Problems'. In: *J. ACM* 45.3 (May 1998), 501–555 (cit. on p. 47).

[54] M. X. Goemans and D. P. Williamson. 'Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming'. In: *J. ACM* 42.6 (Nov. 1995), 1115–1145 (cit. on p. 47).

[55] S. A. Khot and N. K. Vishnoi. *The Unique Games Conjecture, Integrality Gap for Cut Problems and Embeddability of Negative Type Metrics into ℓ_1*. 2013. arXiv: 1305.4581 [cs.CC] (cit. on p. 47).

[56] S. Khot et al. 'Optimal inapproximability results for MAX-CUT and other 2-variable CSPs?' English (US). In: *SIAM Journal on Computing* 37.1 (2007), pp. 319–357 (cit. on p. 47).

[57] P. Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17 (2020), pp. 261–272 (cit. on pp. 53, 75).

[58] M. Born and V. Fock. 'Beweis des Adiabatensatzes'. In: *Zeitschrift fur Physik* 51.3-4 (Mar. 1928), pp. 165–180 (cit. on p. 62).

[59] B. D. M. Jones et al. *Optimising Trotter-Suzuki Decompositions for Quantum Simulation Using Evolutionary Strategies*. 2019. arXiv: 1904.01336 [cs.NE] (cit. on p. 63).

[60] R. Gehrke and K. Reuter. 'Assessing the efficiency of first-principles basin-hopping sampling'. In: *Physical Review B* 79.8 (2009) (cit. on p. 72).

[61] V. Uthayamoorthy. *Variational-Circuits-and-Neural-Networks*. Version 1.0.0. June 2022 (cit. on pp. 74, 77).

[62] J. Spall. 'Multivariate stochastic approximation using a simultaneous perturbation gradient approximation'. In: *IEEE Transactions on Automatic Control* 37.3 (1992), pp. 332–341 (cit. on p. 89).

[63] K. Weinberger. *Lecture 12: Bias-Variance Tradeoff*. 2017 (cit. on p. 98).

[64] P. Tamta, B. P. Pande and H. S. Dhami. *A Polynomial Time Solution to the Clique Problem*. 2019. arXiv: 1403.1178 [cs.DS] (cit. on p. 103).

[65] E. B. Wilson. 'Probable Inference, the Law of Succession, and Statistical Inference'. In: *Journal of the American Statistical Association* 22.158 (1927), 209–212 (cit. on p. 104).

Viroshaan Uthayamoorthy

NTNU
Norwegian University of
Science and Technology

SINTEF