Quantum-enhanced Supervised Learning with Variational Quantum Circuits

Sevak Mardirosian

Leiden University & IBM Netherlands

Leiden Institute of Advanced Computer Science

First Supervisor: Vedran Dunjko
Second Supervisor: Alfons Laarman
Company Supervisors: Cor van der Struijf, Zoltán Szlávik & Benjamin Timmermans

Master's Thesis in Computer Science

July 20, 2019

IBM Nederland B.V. authorizes that this thesis will be made available for inspection by placement in libraries. For publication, for the whole or / and part of this thesis, prior approval needs to be granted by IBM Nederland B.V. which may be only withheld within 90 days and if confidential information is published. During this period the thesis may only be issued to an employee of the institution on which the author of this thesis is currently following a course.

Date: 5/2/2019

Signature .....
IBM mentor

(+ IBM mark)

Abstract

Quantum machine learning (QML), the intersection of quantum computation and classical machine learning, has become a matter of interest over the past few years. Quantum computers, which exploit the laws of quantum theory and utilize properties such as superpositions and entanglement, promise to provide more efficient ways to solve complex computational problems and to analyze big data. In this thesis, we study hybrid classical-quantum algorithms, specifically using so-called variational quantum circuits for a supervised learning binary classification task. The purpose here is to show how near-term quantum devices open up a new avenue to combine quantum computing with classical machine learning methods, to achieve new quantum-enhanced classifiers.

We study the approach of [1] where the input vector is encoded by a quantum feature map, where the data is provided in the conventional way i.e, classically. Afterwards a short-depth quantum circuit is applied to the feature state for training and classification on practical datasets. In the original work, the authors have studied the performance of such a quantum classifier system using devices with 2 work qubits. In this work, we investigate the performance of the system as more qubits are utilized. Using variational quantum circuit, our model was able to classify MNIST dataset up to 90% in accuracy on testing set using 3 qubit systems, which is better than for the case of 2 qubits. While this is a promising result, our study also highlights a number of issues that will have to be dealt with (e.g. training becomes very demanding), before real-world quantum advantages can be exploited.

In memory of

Werner Koster

Contents

Chapter 1

Introduction

Machine Learning (ML), a sub-discipline of artificial intelligence, has been an increasingly important domain of computer science research for more than two decades. The idea behind machine learning is to allow computers to learn from data and therefore help us solve various problems without being explicitly programmed. Machine learning application examples are such as stock market prediction [2; 3], weather forecasting [4] and many others [5; 6].

From 2010 onwards, companies started deploying ML applications at a much higher rate than in previous years. One of the well-known ML applications is Google's Translate engine, which was initially based on statistical machine translation methods. In 2016 Google re-wrote the engine to neural machine translation techniques that uses large artificial neural networks (ANN) to predict the likelihood of a sequence of words and sentences [7] making it much easier for the engine to learn how to translate and to progress. However, with the onset of big high-dimensional datasets [8], we have started encountering computational bottlenecks; modern datasets are becoming too large and too complex to handle with conventional computers[1] and methods. A boost in computing power may come from new machines based on laws of quantum theory, which have been in development by scientists over the last few decades. These machines are called *quantum computers*. Due to their potential, quantum computing has been gaining traction not just in terms of being a

[1] In this thesis we will refer to machine learning algorithms which run on classical computers (laptop desktop or mobile), as classical machine learning.

research topic, but also private and public funding sources [9], aiming to further develop these computers. We will explore quantum computing in the subsequent chapters. One main idea behind quantum computing is the promise of a significant increase in computational speed[1] compared to classical computers. Quantum computers utilize properties of quantum mechanics such as superposition and entanglement, which, intuitively, allows quantum particles to perform a multitude of calculations at the same time. Note, this image provides intuition, however, the real source of quantum computing power is more complicated and not fully understood. This allows us to say that, for a given machine learning task, quantum computers may offer ways to outperform classical computers. On the other hand, machine learning methods have also become a go-to solution for hard computational problems. It is therefore expected that both worlds, classical machine learning and quantum computers will play a significant role in modern computing. The intersection between classical machine learning and quantum information world is called *quantum machine learning*[2] [10]. In this thesis we investigate the potential of quantum classifiers for binary classification in supervised learning. Our quantum classifier, adopted from [1], performs in a similar vein as the Support Vector Machine, in that it finds an (approximation of the) optimal cutting hyperplane. The new quantum classifiers introduce new, previously unexplored parameters, such as the qubit number of the quantum system. We will also utilize approaches such as Principle Component Analysis (PCA) to reduce number of dimensions a datasets has and Cross-Validation techniques for a better model(s) performance estimate. The main research question of this thesis is: how do the new types of model parameters, such as the qubit number, influence the classification performance. Understanding this will be an important step of identifying the best applications of these novel quantum technologies.

[1] More on this in the subsequent chapter.

[2] In short, quantum machine learning summarises approaches in which both, the quantum information world and the classical machine learning world are combined and used to solve a problem or explore a new paradigm.

1.1 Background & Motivation

One general goal behind machine learning techniques is the idea of generalization, that is the ability of a model to properly adapt, or be effective, to new previously unseen data, stemming from the same distribution as the one used to train the model [11]. Machine learning algorithms have been successful and effective across many applications, and often outperform humans for example in board games [12] and image recognition [13]. However, in certain domains [8] the increase of the complexity and dimensionality of data, makes it more difficult to truly reap the benefits of machine learning. The concept of Quantum Machine Learning is gaining traction due to the expectation that such systems could be the answer to complex, machine learning problems [14; 15]. In 2012, Physicist John Preskill [16] first coined the term of *quantum supremacy*, capturing the idea that quantum computers will soon able to execute tasks or solve problems that classical computers cannot. In order to achieve this, quantum computing scientists need to provide evidence or super-polynomial speedups over their best classic counterparts. There are number of quantum algorithms that provide evidence: quantum algorithms providing significant speed-ups over their best classic counterparts. for quantum supremacy including Shor's algorithm. In 1994 Peter Shor invented a quantum factorization algorithm which provides a super-polynomial speedup over the best known classical algorithm [17]. The same applies for Grover's search algorithm by Lov Grover [18], which is a quantum search algorithm that is capable of finding target elements in an unstructured database of N elements using an average of $O(\sqrt{N})$ steps, that constitutes a quadratic speed up over the best possible classical solution at $O(N)$ steps. Moreover, researchers at different companies have declared their intent to prove quantum supremacy, this includes Google [19] and IBM [20]. The current objective of both companies is to increase number of qubits needed for quantum supremacy.

While most of the big companies are eventually targeting full-scale quantum computers, progress and breakthroughs have recently been made in the domain of small-scale devices. The era that we are entering is sometimes called the noisy intermediate scale quantum (NISQ) computing era [16]. Most of these devices have a limited number of qubits and

do not allow error correction. With these small-scale devices, we can test advantages of quantum computing to not only give us new experiment-driven research topics, but also to provide more tangible results of what quantum computers may bring, when applied to problems which may be very difficult for classical computers. The focus has been to find computational problem(s) that can solved by small-scale devices.

Recently the term quantum machine learning attracted a increasing amount of research. For instance, Lloyd et. al [21], provided one of the first quantum machine learning algorithms in supervised and unsupervised learning. The novel developments in quantum machine learning were followed up by a monograph by Wittek [22] titled "Quantum Machine Learning—What quantum computing means to data mining", where he reviews a number of concepts from both worlds and summarises some of the early papers. From 2013 onwards, the number of research papers on the topic has dramatically grown, witnessed by a collection of review papers on the topic [10; 15; 23; 24]. In the same period, a new breed of hybrid classical-quantum algorithms started to emerge. Variational quantum circuits (also referred to as parametrized quantum circuits) were introduced by Peruzzo et al [25] as a member of hybrid quantum-classical techniques. Variational circuits are unitary transformations - thus, processes that can be run on a quantum computer - which depend on a set of tuneable parameters (θ). In other words, we have quantum devices which have tuneable parameters that allow changes to the specifications of an otherwise fixed computation. The key insight is that the circuit parameters can be adjusted until the variational circuit produces the desired output.

Quantum machine learning has several approaches. One approach is concerned with supervised learning, which will be the main focus of this thesis. In supervised learning one has access to dataset with labels, which can be thought of as solutions to a problem. A common example are pictures of cats and dogs, that are labeled as cats and dogs (the problem is then: is a given picture a picture of a cat or of a dog?). In general supervised learning, the learning algorithm learns from data which are independent and identically distributed training example sets $\{(x_1, y_1), \ldots, (x_M, y_M)\}$, where $x_i \in \mathbb{R}^N$ are the data points, and $y_i \in \{-1, 1\}$ are binary labels specifying the class to which a data point be-

longs. N represents number of features i.e., the dimensionality of the data vector and M is the number of training instances. Here, the data in the training set is already classified and the objective is to classify new - unseen - data. One famous example of supervised learning algorithms is the support vector machine algorithm, where the separation of data points is achieved by using a hyperplane separator. Classical computers can find such an optimal classifier, that is, finding the maximum margin separating hyperplane in time $O(log(\epsilon^{-1})(poly(N, M)))$ [23]. Thus, computational limitation arises when the dimension become very high. With a quantum computer this task can become more efficient, mainly because quantum computers have the ability to perform certain computations much faster.

In [23] the authors have proven that a variant of the quantum support vector machine can be implemented with $O(log(NM))$ run-time on both training and classification, when the data is given in a quantum form. However when the data is provided using classical computers, then the methods of [23] cannot be applied. In [1], IBM researchers proposed a family of feature maps for classical data encoding and an SVM type classifier that processes the data provided classically. The classifier we are going to focus on in this thesis report is the *quantum variational classifier*, which operates using a variational quantum circuit to classify a training set in direct analogy to conventional SVMs. These methods do not speed up conventional techniques, in the same way as [23], but may provide means to use "quantum classifiers" which have different generalization performance than possible for a classical computer.

In this thesis we will explore the quantum variational classifier in the context of supervised learning.

1.2 Research Question

Our goal in this thesis is to explore the use of variational circuits as classifiers, to study their performance on real datasets, and to see how new quantum model parameters influence the supervised learning performance. In other words the specific research question

of this thesis is:

> **How does using a larger (more qubit) shallow quantum circuit(s) improve the classification performance on real datasets**?

As increasing the qubit number increases the optimization space, we will also explore the subsidiary question: How does the number of qubits used in the quantum classifier affect the required number of optimization (training) steps?

1.3 Overview

This thesis report is structured in five parts. In Chapter 2, we briefly discuss the fundamental concepts of quantum computing. In Chapter 3, we introduce supervised learning, classification, and learning models in general. In Chapter 4, we introduce concepts adopted from [1] including quantum feature mapping for encoding classical information into quantum state and variational circuits for classification purposes. In Chapter 5, we present our results and experiments and provide a discussion of our findings. Finally, we conclude the thesis in Chapter 6.

Chapter 2

Quantum Computing

In this section we briefly introduce quantum computing in contrast to classical computers. In Section 2.1 we discuss the unit of information in quantum computers. In Section 2.2 and 2.3 we discuss quantum gates and quantum circuits, respectively. Finally, we briefly introduce IBM's quantum experience platform, which we use to program quantum computations.

2.1 Qubits

Classical computers, which are made of transistors, function by manipulating bits which are pieces of data. The unit of information is a *bit* and the state (or value) of a classic bit is described by the values 0 and 1. In the quantum world, the quantum computation is based on an analogous concept, referred to as quantum bit or qubit, which can be in not only the discrete quantum states $|0\rangle$ and $|1\rangle$, but also in a superposition the two states.

In quantum computing, the state of a qubit is a unit vector in a two-dimensional complex vector space. Consider a two dimensional quantum system, specified by the so-called *computational basis* states $|0\rangle$ and $|1\rangle$. These basis states are numerically represented with the column vectors $\begin{bmatrix}1\\0\end{bmatrix}$ and $\begin{bmatrix}0\\1\end{bmatrix}$, respectively. This forms an *orthonormal* basis for the qubit vector space (known as the Hilbert space).

The main difference between classical bits and quantum bits is that quantum bits can be in a different states other than $|0\rangle$ or $|1\rangle$. A qubit can be in both states simultaneously (a superposition of two states), for example

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

When a qubit is measured in the computational basis, the result will yield either a state $|0\rangle$ with a probability equal to $|\alpha|^2$ or a state $|1\rangle$ with a probability equal to $|\beta|^2$. α and β are called probability amplitudes such that $|\alpha|^2 + |\beta|^2 = 1$.

More generally, for a register of n qubits, each of which can be in any of the two basis states, the general quantum state can be in any superposition (linear combination) of the 2^n possible bit configurations, so in the state $|\psi_n\rangle = \sum_{b_1,\ldots b_n} \alpha_{b_1,\ldots b_n}\,|b_1,\ldots b_n\rangle$, where $b_j \in \{0,1\}$, with the normalization constraint $\sum_{b_1,\ldots b_n} |\alpha_{b_1,\ldots b_n}|^2 = 1$. The numerical representative of this state is the exponentially long vector of the quantum amplitudes $\alpha_{b_1,\ldots b_n}$. The exponential size of this vector is one of the key bottlenecks in the classical simulation of quantum computations. Given such a state, the probability of measuring (observing) any of the bit-configuration $(b_1,\ldots b_n)$ is given by $P(b_1,\ldots b_n) = |\alpha_{b_1,\ldots b_n}|^2$.

2.2 Gates

To manipulate a register of quantum bits, and thereby its quantum state, we use quantum gates. There are a number of gates for both single-qubit and multi-qubit gate systems. One commonly used quantum gate operation is the Hadamard gate H. The Hadamard gate acts on a single-qubit and it is represented as:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.2}$$

By Applying H gate to $|0\rangle$ we change the state of the qubit to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Measuring it in computational basis yields 0 with probability $\frac{1}{2}$ and 1 with probability $\frac{1}{2}$. Below, we

have an example of such an operation.

$$H \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.3}$$

Which is the state

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}. \tag{2.4}$$

There are other gate operations such as X-gate, phaseflip gate Z operation and CNOT operation. The latter is a multi-qubit gate operation. Below we give the unitary matrix representation of CNOT:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In general, single qubit gates can also be parametrized by real parameters. For instance the $Z(\phi)$ gate is given by

$$Z(\phi) = diag(1, exp(i\phi))$$

where $diag(a, b)$ represents a diagonal matrix with diagonal elements a, b, and the overall matrix represents a single qubit rotation around the Z basis, by the angle ϕ.

2.3 Circuits

A quantum circuit is a collection of gates that are applied on a register of n quantum bits.

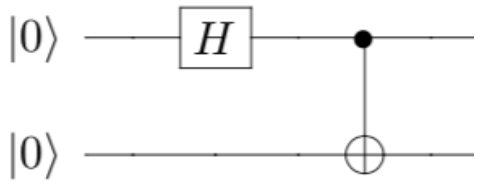


Figure 2.1: A simple quantum circuit

In Fig. 2.1 we have a quantum circuit diagram. Each line depicts a qubit. The initial value of a qubit is usually $|0\rangle$, but this is not always true. In Fig. 2.1 we have a circuit with two qubits and two gates. The first gate applied on the first qubit is an H gate. By applying H gate we create a quantum superposition in the illustrated case $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This is followed by a CNOT gate we mentioned earlier. A CNOT is applied on both the first and second line making the first qubit a *control qubit* and the second a *taget qubit*. CNOT gate can be used to create an entangled state. An example we have in Fig. 2.1. The circuit we have here creates an entangled state for each state of the computational basis $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ which we input to the circuit. We will illustrate the effect of this circuit shortly in more detail.

In general, quantum mechanics allows all unitary operations, which are in general specified by $2^n \times 2^n$ complex unitary matrices. Recall a rectangular matrix U is unitary if its conjugate transpose $U^\dagger$ is its inverse, so $UU\dagger = U^\dagger U = I$. In quantum computing, we utilize quantum circuits to construct important subsets of all possible unitary operations, much like boolean logic gates (AND, NOT) are used to construct arbitrary boolean functions in classical computing. It can be shown that single qubit Z-rotations, Hadamards and CNOT gates we introduced suffice to generate all unitary transformations, much like AND, NOT and fan-outs suffice for all classical boolean functions [26]. It is important to note that applying even local single qubit gates on a n-qubit quantum system can require the update of all of the exponentially many amplitudes specifying the final quantum state. This exponential cost of representation is one of the reasons simulating quantum computation may take exponential time and space in the number of qubits of our quantum computer. Of particular interest to this thesis are so-called *short-depth quantum* circuits,

that is, quantum circuits which are of constant depth. They correspond to the type of computations we may expect to be achievable by real-world quantum computers in the near-term.

Another essential operation in quantum computing is *measurement*. A qubit can be in different states, but when measured the quantum bit in state $|\psi\rangle$ collapses and is now basically a classical bit with a state of either 0 or 1. In other words, measurement destroys the information about the superposition of a quantum state - that is, the values of α and β - of a quantum bit. However, critically, as explained earlier, the values of the amplitudes affect the probabilities of individual measurement outcomes. Circuit symbol for measurement represented in Fig. 2.2.

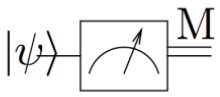


Figure 2.2: Measurement circuit symbol representation

Although quantum mechanics allows quantum measurements relative to any basis, we will focus only on the so called Z-basis measurements which "project" a quantum register of qubits to one bit string configuration, with probability given by the amplitude-squared of the bit string configuration in the quantum state, as explained earlier.

2.4 IBM Q Experience

IBM researchers developed an online platform called IBM Q Experience. Here, one can program and experiment with a real quantum chip. The interface is easy to use and quite self-explanatory. To start, one must create a circuit. Afterwards, one can start adding gates by dragging it to the circuit, as shown in Fig. 2.3.

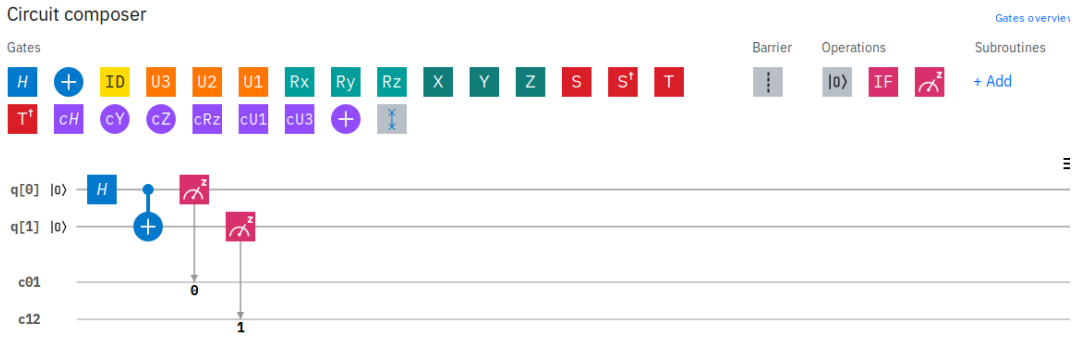


Figure 2.3: IBM quantum composer

Here, we have rebuilt our representation in Fig. 2.1, which consists of 2-qubits and both are initialized to $|0\rangle$. We apply H gate on the first qubit, which creates a uniform superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Next, we apply the CNOT operation such that the second qubit is *conditionally* flipped, dependent on the state of first qubit. If the state of the first qubit is $|0\rangle$, then no action would occur on the second qubit and result is just the state $|0\rangle\,|0\rangle$. However, if the state of the first qubit is $|1\rangle$, then a NOT would be applied to the second qubit and the register would be in the state $|1\rangle\,|1\rangle$. By the superposition principle, then if we prepare the first qubit in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, the output is the entangled state $\frac{1}{\sqrt{2}}(|0\rangle\,|0\rangle + |1\rangle\,|1\rangle)$.

Furthermore, IBM researchers developed an open-source library called Qiskit (quantum information software kit)[27] written in Python scripting language, which offers the possibility to either use local quantum simulators or REST API[1] to deploy jobs on a real quantum chip developed by IBM. Qiskit consists of different components[2]. Every component has its own set of tools one can use for different purposes.

[1]https://en.wikipedia.org/wiki/Representational_state_transfer
[2]More about, please follow the link https://qiskit.org/

Chapter 3

Classical Machine Learning

Machine learning has become an important field in computer science and also in our daily lives. Nowadays, many of the devices we interact on a daily basis have a learning algorithm behind it. Research in this field is growing incredibly fast and lots of applications are being developed and deployed almost on a daily basis. In this section we will talk about a sub-field of machine learning and that is *Supervised Learning*. Given the amount of topics this category has, we will focus only on a small subset of topics relevant for our project.

In the subsequent sections we will explain how supervised learning works in general and then move on to models and specifically the support vector machine. Further, we will talk about optimizers and briefly talk about high dimensionality techniques and reduction of this to fit number of qubits we have[1].

3.1 Supervised Learning

One of the tasks in machine learning is supervised learning. In supervised learning the objective is to learn from labelled training data to make future predictions on unseen data.

This mode of learning is called *supervised* because correctly labeled examples are pro-

[1]This is described in Section 3.6

vided to the learning model. Here, the algorithm receives a set inputs along with known outputs (we will refer to input data as data-points and data-vectors, and to the labels as outputs, interchangeably), which then tries to predict a label based on features that the learning algorithm distilled through the training process. One example of supervised learning is of e-mail spam filtering, where we train a model on a labeled dataset - emails that are marked as spam or not-spam - to predict if a new incoming email is spam or not. This is a typical classification task, where a supervised learning algorithm predicts a discrete *class labels* i.e., a category the data belongs to. Classification with two labels is called *binary* classification, and in this thesis we will be exclusively dealing with this case. There are number of algorithms that can be used to solve classification problems such as Neural networks (NN), K-Nearest Neighbor Classifier (KNN), Decision Tree, Naive Bayes, Logistic Regression and Support Vector Machine (SVM) [28] [1], which will be the focus this thesis.

3.2 Overfitting and underfitting

Overfitting and underfitting are two common problems in machine learning in general. Overfitting refers to the settings where the learning model performs (very) well on the training set, but fails to generalize this performance to unseen (future) data. It often arises in noisy scenarios when the learning model is complex relative to the true nature of the noise-free training data. In this case the model can learn (or specialize to) the noise, rather than the data features themselves, yielding poor generalization performance.

Underfitting, the opposite of overfitting, occurs when the model is either too simple (performs poorly) for the defined problem or unable to draw patterns or learn from the data. One possible solution is to select a more powerful model i.e., a model with more parameters. In Fig. 3.1 the problem of overfitting and underfitting illustrated using a more complex, nonlinear decision boundary.

[1] (explained in Section 3.4).

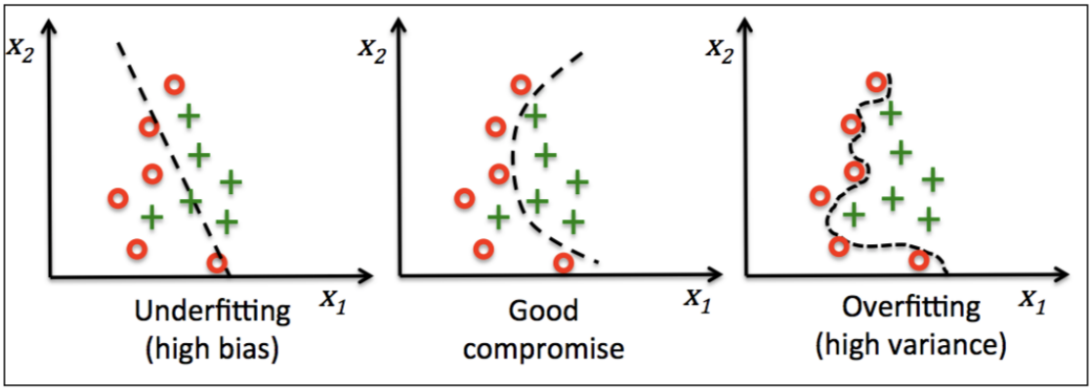


Figure 3.1: Overfitting & Underfitting in machine learning models [29]

It is important to avoid both problems. Often the only way in practice that allows us to verify whether the learning model generalizes well to new unseen data is to test it. For this, a technique such cross-validation is recommended as it is good way to test the generalization performance of the model.

3.3 Cross-Validation

Generalization is one of the key steps in correctly using a machine learning model that invokes the estimates of the performance of the model on unseen data. For example, if we fit[1] a model on training data and use the same data to test our model, the performance will either suffer from underfitting or overfitting. Note that such testing cannot be done on the training data itself.

To obtain reliable estimates on how well a model will do on a class of data, *cross-validation* technique and specifically k-fold cross-validation is used. k-folds cross-validation splits the training dataset into k subsets, called k-folds, where, where k-1 folds are used for the model training and one fold is used for testing. This results in k model generalization performance estimates. We then calculate the average performance of each model, which is based on the independent folds, this to obtain model performance that is less sensitive than a single performance estimate on the entire training set. Figure 3.2 summarizes the

[1]The terms "fit", "train" and "learn" a model are often used interchangeably.)

concept of k-fold cross-validation where $k = 10$.

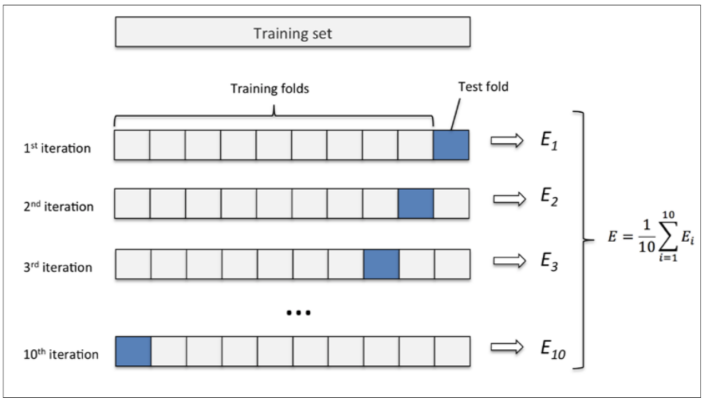


Figure 3.2: k-fold cross-validation - figure from [30]

In Figure 3.2, the training set is divided into 10-folds. Nine folds are used for training and 1-fold is used for testing and model evaluation. The estimated performances E_i, is used to calculate the estimated average performance E of the model.
Note that the standard values for k-fold cross-validation are between 5 and 10. This number of course depends on how large the dataset is and the number of instances it has. For small sets 10-folds is recommended [31].

3.4 Support Vector Machine

Support Vector Machine (SVM) [28], is a supervised machine learning classification algorithm that constructs a separating hyperplane for classification. SVMs are typically used for binary classification, and we will often refer to two binary labels in $\{0, 1\}$ in this section. The objective behind SVM is to maximize the *margin*. Margin is defined to be the distance between the separating hyperplane and the training samples (of both 0 and 1 instances) that are closest to this hyperplane.

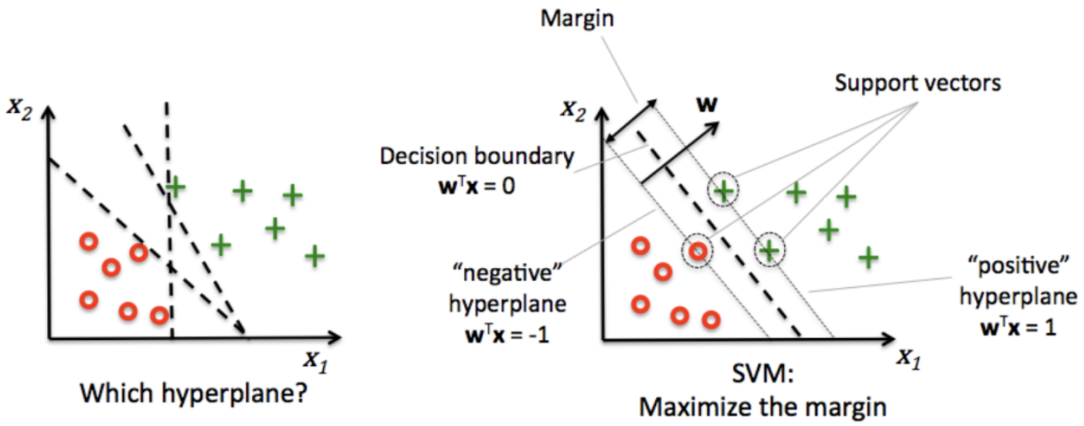


Figure 3.3: A hyperplane for separating 2-dimensional data - figure from [32]

Having decision boundaries with large margins yields a low generalization error - due to poor training performance, whereas decision boundaries with small margins tend to be more prone to overfitting. Our goal is to find the best separating hyperplane with maximum margin. Hyperplanes are in general defined with:

$$\vec{w}^T\vec{x} + b = 0 \tag{3.1}$$

where b is the off-set, $\vec{w}$ is the normal vector, $\vec{x}$, and $\vec{x}$ the data vector so:

$$\vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.2}$$

The SVM classifier is then defined with

$$y_i = f(x) = sgn(\vec{w}^T x_i + b) \tag{3.3}$$

The goal of training an SVM is to find the parameters w and b which specify the maximal margin hyperplane, see [28; 32; 33] for more details on how this can be done.

3.4.1 Kernel-based SVM

Often, the training data will not be linearly separable. Not being linearly separable simply means that no hyperplane can correctly separate samples to classes. In SVM this problem is solved by applying a non-linear transformation to the input data, which maps the data to a higher dimensional so-called feature space[1]. Representing data in a higher-dimensional space $\phi(x)$, where ϕ is a suitably chosen non-linear function, makes it more likely that the mapped data points will be linearly separable in the feature space, see Fig. 3.4 for a simple example.

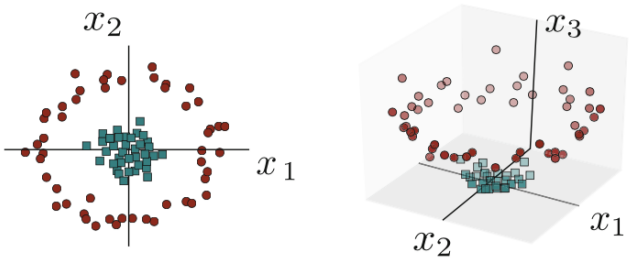


Figure 3.4: Data points of two classes which cannot be separated by a hyperplane (left), However, if we project non-linear data points into higher dimension (3 dim, in this case), then we can find a hyperplane in the feature space (right) - figure from [34]

The idea to use non-linear combinations of the original features to generate easily separable representations is central to so-called kernel methods such as SVMs. Also, the key point behind kernels is that to actually find the optimal classifiers, i.e., to train the model, we only need to be able to compute dot product of the mapped points, defined with $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. The function K, is then called the kernel function. So, as long as the kernel function can be computed without explicitly mapping $x_i \mapsto \phi(x_i)$, we can avoid having to deal with high-dimensional representation. Thus, perhaps the most interesting property of kernel functions in SVMs is that there is no need to explicitly compute the feature maps, as long as the kernel functions can be computed for all pairs of data points. This can lead to significant training speed-ups, and in fact, allows us to

[1]There are other methods which employ soft-margins, and allow for error in classification, but we will not be discussing those here.

use feature maps with formally have infinitely-dimensional feature spaces such as those reached by the so-called radial basis functions.

3.5 Cost Function in Learning

A cost function (also called a loss function), provides a distinctive measure of the performance of our model on a given data point. Simply stated, the loss function is a function that measures the distance from the model's estimated value (output or label) to the real expected value. In supervised learning, the value of the cost function on a point is referred to as loss, and it measures how far the model predictions are from the target labels. A third closely related term is the accuracy of the model for on both the training and testing, which is essentially the averaged loss over the entire dataset. We will discuss this in more detail shortly. In the majority of machine learning techniques what we are trying to do is to minimize the cost function for better results. This shows how right or wrong the model is when predicting an outcome, which depends on the learning stage. After training process, we evaluate a model on a separate set of test data with respect to the same cost function.
In this work we will adopt the cost function defined in [1], which is essentially the averaged number of missclassifications, and explained in Section 4.2.1. Other possibilities include the cross entropy [35], which is particularly well suited when the output is a sample according to some probability distribution.

3.6 Dimensionality Reduction

The datasets used for machine learning and statistics are growing in size, becoming high in dimension i.e., high in number of features, and are usually noisy. High dimensionality usually causes problems which is known as the *curse of dimensionality* [36]. The curse of dimensionality is referring to a set of phenomena that occurs when dataset is high-dimensional. This has unfortunate consequences on the behavior and performances of learning algorithms. One problem is *overfitting* the dataset - the model has a low power

of generalization - and also, performance degradation of the model.

One of the key aspects of the curse of dimensionality is the fact that the search space grows exponentially with the number of dimensions. That is, to characterize even a low-dimensional manifold (which is embedded in a high dimensional space), the number of points needed to describe it scales exponentially with the embedding space, even if the manifold dimension is low [37].
A good method to resolve this problem is to reduce number of dimensions, but the downside of this method is the potential loss of valuable information. One such method is **Principal Component Analysis** (PCA). PCA reduces the number dimensions of the data, by linearly transforming the features to uncorrelated weightings of the original features (feature components). This is done in a manner which attempts to maintain the full variability of the data, so still allowing for the identification of the relevant patterns [38; 39].

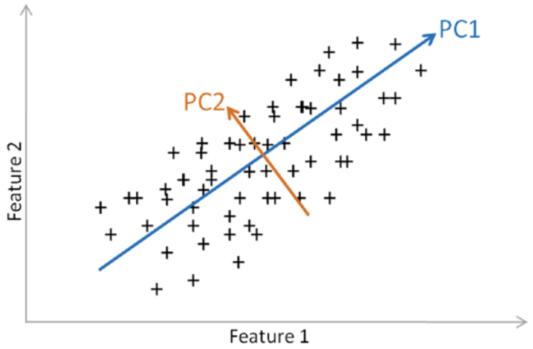


Figure 3.5: PCA finds the maximum amount of variance in in one-dimensional directions [38]. Keeping the data variance ensures that the quintessential patterns in the data are preserved even though the dimensionality is (dramatically) reduced.

PCA uses methods like singular value decomposition to find a linear transformation to a new set of coordinate axes (a new basis of the vector space), such that the variance along each of the new axes is maximised (see Fig 3.5). For our purpose, we used PCA as a pre-processing step to reduce number features a dataset has to match number of qubits our circuit has.

3.7 Optimization

Optimization methods are one of the fundamental techniques used in machine learning algorithms to find the best parameters for a given model. Optimization problems consists of an objective that needs to be either maximized or minimized and parameters that can be tuned.

In this thesis we will be using a particular class of optimizers, motivated by the nature of quantum computers. As we discussed earlier, quantum computers are fragile and more more error-prone than their classical counterparts. By performing quantum computations quantum computers undergo so-called *quantum decoherence*, a phenomenon where the environment perturbs the quantum state, leading to error, or noise, as it is more generally referred to. When quantum decoherence occurs, qubits lose information to the environment over time and because of this it destroys the results of the calculation hence causing the qubit to collapse into a classical state [40].

In our work, the classifier we train will be a quantum computation, so itself is prone to decoherence and resulting noise and errors.

One of the optimizers recommended for noisy experimental settings is simultaneous perturbation stochastic approximation (SPSA) [41]. SPSA is a global optimizer that is based on stochastic gradient descent algorithm and minimizes the loss function by following the gradient method. In short, the objective of stochastic gradient descent (SGD) is to iterate a weight update based on the gradient of the loss function. SPSA comes with a number of features and methods that are designed to find the global minima. In Section 2.4 we briefly discussed Qiskit. One of the many features Qiskit offers is optimization. By simply importing optimization modules, one is able to use many of the already implemented optimizers.

Chapter 4

Learning with Quantum Circuits

In the previous section we discussed classification in machine learning and how to use linear and non-linear SVM methods to solve certain problems. However, limitation of this method starts when the feature space becomes large, and kernel functions become hard to estimate.
In this thesis we focus on one of the two quantum variational classifiers developed in [1], and which builds on [42; 43]. Here, a variational classifier is used to generate a separating hyperplane in the quantum feature space, which may be high-dimensional and difficult to work with on a classical computer, yet tractable for a quantum machine.

In this chapter we present the techniques proposed in [1]. Specifically, encoding classical information into a quantum state, called quantum input state preparation. Input state preparation could be a big challenge for achieving a quantum advantage with existing quantum machine learning algorithms. Then, we discuss how short-depth quantum circuits can be used for training and classification. In the end we will discuss how artificial data for initial testing of the resulting quantum classifier is generated.

4.1 Quantum Feature Mapping

The key idea of quantum SVM are kernels which distinguish it from classical SVM kernels (such as Gaussian or Polynomial Kernels). Quantum feature mapping function maps classical data points, i.e. input $\vec{x}$, non-linearly to a quantum state, i.e. realizes the map:

$$\phi : \vec{x} \mapsto |\phi(\vec{x})\rangle \langle\phi(\vec{x})| . \tag{4.1}$$

The authors of [1] undertook a balancing act: the construction of feature maps whose mathematical properties are on one hand complex enough so that data could be easily separable after their mapping, and yet simple enough to be implementable on a real quantum processor. One of the feature maps we adopted in our implementation is defined via the unitary:

$$U_{\Phi(x)} = \exp\left(i \sum_{S \subseteq [1,n]} \phi_S(\vec{x}) \prod_{i \in S} Z_i \right) \tag{4.2}$$

Where $S \in \{0, 1, ..., n-1, (0,1), (0,2), ..., (n-2, n-1)\}, \phi_i(\vec{x}) = x_i, \phi_{(i,j)}(\vec{x}) = (\pi - x_i) * (\pi - x_j)$ [44]. Note, the mapping ϕ from 4.1 is a classical to quantum map, whereas each of the ϕ_i maps simply transform the individual coordinates. This notation was used in [1]. $\mathcal{U}_{\phi(\vec{x})}$ is a feature map function defined on n-qubits, whose dimension is exponential in the data dimension and qubit number, generated by the unitary:

$$\mathcal{U}_{\phi(\vec{x})} = U_{\phi(\vec{x})} H^{\otimes n} U_{\phi(\vec{x})} H^{\otimes n} \tag{4.3}$$

which maps classical data input $\vec{x}$ into a quantum state. More specifically, to realize the mapping ϕ from Eq. 4.2, we apply $\mathcal{U}_{\phi(\vec{x})}$ to a fixed quantum state. So the data points are embedded in the unitary $\mathcal{U}_{\phi(\vec{x})}$ and not the input state. We will also refer to the unitary $\mathcal{U}_{\phi(\vec{x})}$ as "the feature map". Note that the unitary $\mathcal{U}_{\phi(\vec{x})}$ is built by the repetition of the block $U_{\phi(x)} H^n$, and in 4.3 this block is repeated d=2 times. However, to realize more complicated, deeper, feature maps, this quantity can be increased Note that in [1], the authors used d=2 (d is the depth of the circuit $U_{\phi(x)} H^{\otimes n} U_{\phi(x)} H^{\otimes n}$ in the circuit) for their implementation. For our experiments in Section 5, we used the same value for

the feature mapping circuit. So the parameters (d, $\phi(x)$) are tunable for classification algorithm explained in Section 4.2.

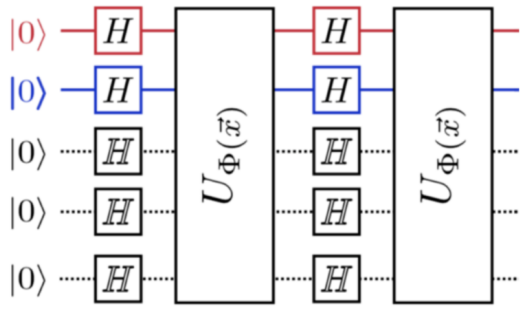


Figure 4.1: Feature mapping circuit - Figure from [1]

The circuit in fig 4.1 for feature mapping consists of two steps, repeated two times ($d = 2$). First, Hadamard gates are applied onto all qubits, then a diagonal gate, which depends on the data.
One of the ideas behind these maps is to need to encode the data in a way classical algorithm(s) will have trouble simulating, thereby having the opportunity to obtain an advantage over classical approaches. The map $U_{\phi(x)}$, for the two qubit case, has a simple form:

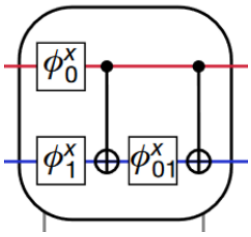


Figure 4.2: U_ϕ details - Image from [1]

where the ϕ gate are just rotations following this rule: $\phi_i(\vec{x}) = x_i$ and $\phi_{i,j}(x) = (\pi - x_i)(\pi - x_j)$. The gates in Fig.4.2 are defined and briefly explained in Section 5.2.2.
Finally, the classical data input $\vec{x}$ is loaded by applying the unitary function to the initial

quantum state $|0\rangle^{\otimes n}$:

$$|\phi(\vec{x})\rangle = U_{\phi(\vec{x})} |0\rangle^{\otimes n} \tag{4.4}$$

Depending on the connectivity graph of the quantum device quantum feature mapping has about $\frac{n(n-1)}{2}$ (n denotes number of qubits) further parameters that can be used to encode more data. After feature mapping, short-depth quantum circuit is applied to the feature state. The output is then trained through multiple layers of parametric gates, by adjusting the parameters (weights).

4.2 Variational Circuit

The overall quantum classifier will consist of two concatenated circuits. The first is the "feature map" circuit, we just discussed. The other, combined with the measurement, effectuates a hyperplane classification in the feature spaces. The parameter of this second circuit are the training parameters of the model we consider. This second circuit is a variational circuit, also referred to as short-depth quantum circuit $W(\vec{\theta})$. A variational circuit is a hybrid quantum-classical circuit [25] which consists of parameterized gates that depends on set of parameters θ as well as an learning algorithm and objective function. In our case, we will train the circuit to correctly label the data-points.
If we use some circuit parameters to feed inputs into the circuit and compute the results (outputs), we can now see that a variational circuit has similar intuition as other supervised learning models. Variational circuit are used for various near-term applications, such as optimization [45] and in context of machine learning, as explored in this thesis. Note that the name variational reflects the variational approach to changing the parameters.

The authors of [1] designed a variational circuit that is composed of l layers is parameterized by $\vec{\theta} \in \mathbb{R}^{2n(l+1)}$ that will be optimized during the training using the classification error on the training set as feedback. Here, l denotes the depth of the variational circuit, i.e. the number of repetitions of so-called entangler blocks (so blocks of two qubit gates

which can introduce entanglement), and single-qubit rotation layers, denoted as $W(\theta)$ and defined as such:

$$W(\theta) = U_{loc}^{(l)}(\theta_l)U_{ent}...U_{loc}^{(2)}(\theta_2)U_{ent}U_{loc}^{(1)}(\theta_1). \tag{4.5}$$

This is presented in Fig. 4.3.

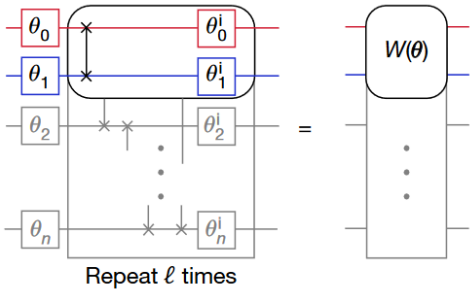


Figure 4.3: Variational circuit - Figure from [1]

As Fig.4.3 shows, each U_{loc} unitary consists of n single qubit unitaries $R(\theta_i)$ [denoted only with the angle in the figure] acting on each qubit. These angles constitute the free parameters. The number of controlled-phase gates is $n+1$ in each entrangler block. More about this in Section 5.2.2.
The single qubit rotations correspond to Z-rotations and Y-rotations and the entangler U_{ent} block consists of controlled-Z gates. This to simplify number of parameters to be handled by the classical optimizer. Practically, this means that we have $2n(l+1)$ parameters to optimize (two per gate). Variational circuits have a good similarity to neural networks, which has weights that we need to optimize, and were referred to as quantum neural networks in the early days of quantum computing.

4.2.1 Training

The variational algorithm designed in [1] consists of two parts: a training stage and a classification stage. For the training, a set of labeled points are given to perform training. We first train the classifier by optimizing the set of parametrizing angles ($\vec{\theta}$). As mentioned before, we utilized SPSA optimizer, which is based on stochastic gradient decent

algorithm, for optimization.

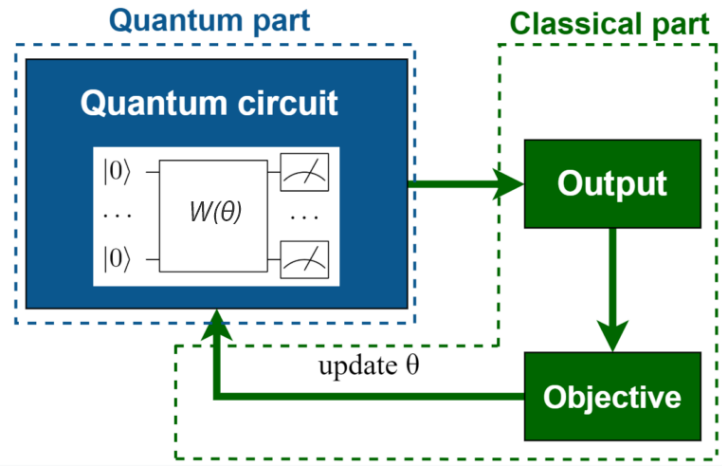


Figure 4.4: The process of variational circuit - Figure from [46]

More specifically, our overall objective is to find an optimal classifying circuit $W(\theta)$ that separates the data sets with different labels. For the optimization we need to define a *cost function* in order to minimize the probability of assigning the wrong label.
In this work we will use the empirical risk $R_{emp}(\theta)$ is defined as a cost function, given by the error probability $Pr(\tilde{m}(\vec{x})) \neq m(\vec{x})$ of assigning the incorrect label averaged over the samples in the training set T

$$R_{emp}(\theta) = \frac{1}{|T|} \sum_{\vec{x} \in T} P_r(\tilde{m}(\vec{x} \neq m(\vec{x})). \tag{4.6}$$

Where $\tilde{m}(\vec{x})$ is the new predicted label and $m(\vec{x})$ is a given output label. We deal with probabilities, as we will explain in more detail later, our classification model is not deterministic.

4.2.2 Classification & Labeling

Next, we describe how our quantum model assigns labels to data-points in more detail. After constructing our variational circuit, we encoded data and now we would like to extract information from it i.e., how we convert quantum information into classical one?

By measuring the circuit. To stabilize the stochastic effects of the measurements, the circuit will actually be measured a number of times, as to estimate the probability of its outcomes, which will then be used to assign the label.
In fact, we will be running the circuit which we describe next some R times (or shots; in Qiskit literature, one sampling from the output distribution of a quantum circuit is referred to a shot, and many shots are needed to estimate the probabilities), and this value R will constitute yet another hyperparameter of the model.

Note that each time we take a shot (run the circuit), we are preparing the quantum circuit from scratch.

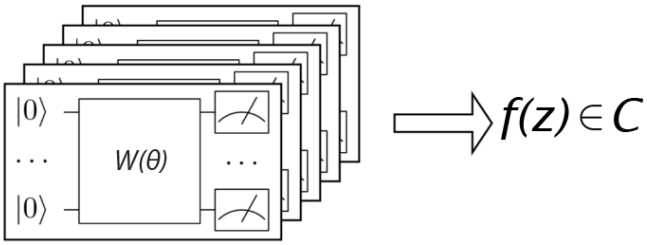


Figure 4.5: Running the quantum circuit multiple times to estimate the expectation value of some chosen observable - Figure from [47]

Specifically, for a two label classification $y = \{+1, -1\}$, a binary measurement characterized by the projection operators $\{M_y\}_{y=0,1}$ is applied to the state $W(\theta)U_{\phi\vec{x}}\,|0\rangle^n$. For a binary classification task, the results are measured in computational basis. The binary measurement is obtained from the parity function $f = Z_1 Z_2$. The results, which are bit strings $z \in \{0,1\}^n$, assigned to a label based on a predetermined boolean function $f : \{0,1\}^n \to \{-1,1\}$. In other words, the classifier is defined by:

- measuring the circuit outcome (which yields a bit string of length n)
- by computing the parity of the bit string, so a measurement consisting of

an even number of '1's is mapped to the first class, and a measurement outcome consisting of an odd number of '1's is mapped to the second class.

Note that the classifier described here is probabilistic ,which it means we might get different outcomes on each run, with the same parameters. The probability of measuring either label $y \in \{+1, -1\}$ is given by:

$$p_y = \frac{1}{2}(1 + y \langle \phi(\vec{x}) | W^T f W(\theta) | \phi(\vec{x}) \rangle \tag{4.7}$$

Where the diagonal operator f is defined as such

$$f = \sum_{z \in \{0,1\}^n} f(z) |z\rangle \langle z| \tag{4.8}$$

And f is a parity function (a XOR of all the bits), yielding -1 for odd, and 1 for even parities as explained. By taking R shots (running the circuits many times) for a single data-point r, we derive the empirical probabilities of assigning the first or second class. For a given training data-point and the corresponding training class label, we calculate a cost value which represents the probability of correct/false classification.

The following steps summarise the procedure of the quantum classifier described above.

1. We combine the two circuits: Variational and feature mapping circuit, we encode data input x using the feature map function defined in Eq. 4.2.
2. Apply a θ-parametrized variational circuit to the feature state $W(\theta)U_{\phi\vec{x}} |0\rangle^n$.
3. Measure the circuit R shots for a binary measurement M_y. Here, using a parity function we assign a class label to the measurement outcome, which are given as a bit strings. After running the same circuit many times, we derive the empirical probabilities of assigning the first or second class.
4. Minimize the circuit error with the defined cost function using the SPSA optimizer.
5. Evaluate the overall performance of the training set and use this as the objective function to train our classifier and update θ accordingly.

4.3 Artificial Data

To check the implementations and evaluate the performance and feasibility of the quantum classifier, the authors in [1] proposed a family of artificial data sets which can be used for initial testing. Specifically, in [1] the model was tested only on artificial data sets. Here we will describe how the artificial data sets are generated for the case $n = 2$, where n denotes number of qubits.

After training, the data should be separated, using the above described feature map by a hyperplane in the quantum state space. For our classifier, we are given a training set T and a test set S of a subset $\Omega \subset R^d$. Both are labelled by a map $m : T \cup S \to \{+1, -1\}$. The labels for the training and testing are generated by choosing a parity function $f = Z_1 Z_2$ and random unitary $V \in SU(4)$. V is applied to the state $\phi(\vec{x})$ and then expectation value $\langle \phi(\vec{x}) | V^T f V | \phi(\vec{x}) \rangle$ of the parity operator $Z_1 Z_2$ is evaluated and is used to assign a label to labelled data point. Afterwards, we map each value to -1 or +1 according to function $f : \{0, 1\}^n \to \{-1, 1\}$. Sampling from the final state returns a value in $[-1, 1]$ for a given x with a constant threshold (separation gap - more on this in Section 5.3) value of $\Delta = 0.3$. We assign label $+1$ if $\langle \phi(\vec{x}) | V^T f V | \phi(\vec{x}) \rangle \geq \Delta$ and -1 when $\langle \phi(\vec{x}) | V^T f V | \phi(\vec{x}) \rangle \leq \Delta$. To provide some intuition behind this dataset, note that if the variational circuit of the quantum classifier is tuned as to approximate the unitary V, then by construction that classifier will correctly label the points. The Δ margin ensures that the stochastic effects do not jeopardize the classification performance nor the training.

Chapter 5

Experiments & Results

In this chapter we provide information about our experimental setup followed by results and analysis of the trade-off between different chosen parameters. Although the quantum SVM is intended to be run on a real quantum computer, due to limited access, the entire study was performed on simulators without any noise. Even in the noiseless case, the simulation times were quite significant, highlighting the need for quantum computers, as the model size increases.

5.1 Experimental Design

In our experiments, we first verified and experimented with our implementations on artificial data described in Section 4.3. Following this we performed experiments on more standard data sets, specifically MNIST, Breastcancer and Wine [48].
The quantum models come with a number of hyperparmeters (described in the next subsection), and we explored the performance of the models as a function of some of them. We tracked empirical risk (training performance), test performance, the difference between the two (generalization performance), and required training times. The key performance category, generalization performance, was evaluated using k-fold cross-validation.

5.1.1 Choice of Parameters

The hyperparameters we chose to tweak and experiment with are:

- Qubits (n) - Number of qubits a circuit has.
- Shots (s) - Number of times a circuit is run to achieve desired output.
- Epochs (e) - SPSA [49] optimizer takes a constant (fixed) maximum number of iterations to perform. Note, also the choice of the optimization method is a hyper-parameter.
- v-depth (l) - Depth of the variational circuit.
- f-depth (f) - Depth of the feature mapping circuit.

Aside from the parameter choice, to fit our data onto the dimensionality the quantum model allows (in our case, the qubit number n) we used dimensionality reduction methods. Although there are a number of options here, we mostly used principal component analysis, and only report on those results here.

5.2 Circuits Implemented

In this section we briefly explain the two circuits (feature mapping and variational) we implemented using Qiskit [27]. The figures below are for a 2-qubit system, and depth is 2 as well. In later experiments we increased n (number of qubits) to 3 and l (depth of the circuit) to 4.

5.2.1 Quantum Feature Mapping

In Section 4.1 we explained the working of quantum feature mapping. Here, we construct our feature mapping circuit. The circuit we implemented is illustrated in Fig. 5.1. Recall that in the quantum classifier, the input vector values are mapped into the parameters

in the unitary. In the example Fig. 5.1 we have used the data points $\begin{pmatrix} x_1 = 1.5 \\ x_2 = 0.3 \end{pmatrix}$, which generated the angles as given in the figure[1].

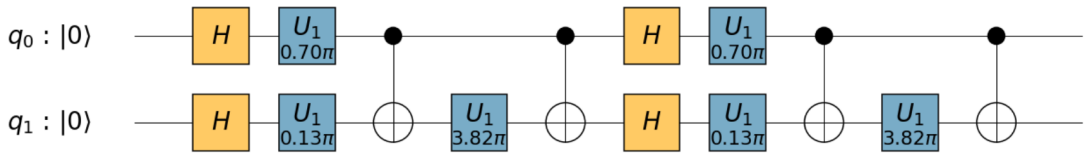


Figure 5.1: Feature Mapping Circuit

As mentioned in Section 4.1, H-gate is initially applied on all qubits. This allows us to prepare an equal superposition of all bitstrings. This is followed by a *controlled-u1* gate, which is the controlled version of the $u1$ single qubit gate, and it is useful as it allows us to apply a quantum phase and its defined with:

$$u1(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix},$$

Furthermore, a multi-qubit gate *controlled-NOT* is applied. Recall controlled-NOT gate flips the target, when the qubit state is in $|1\rangle$. Controlled-NOT is defined as such:

$$C_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The purpose of this circuit is to encode classical data into a quantum state. Thus, the circuit in Fig. 5.1 maps classical input datapoint $\vec{x}$ to the quantum feature space whose dimension is exponential in the number of qubits. As we explained in detail earlier, afterwards a short-depth quantum circuit (discussed in subsequent section) $W(\vec{\theta})$ is applied

[1]Note: For illustration purposes we used $u1$ gate instead of rz gate. Both gates have the same effect.

to the feature state.

5.2.2 Variational Circuit

We also talked about variational ciruits in Section 4.2 and what purpose it serves. Here, we translate Eq. 4.5 into Python code using Qiskit [27]. The results are presented in Fig. 5.2.

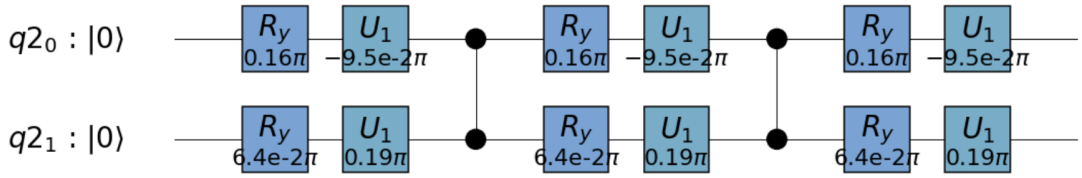


Figure 5.2: Variational Circuit

In the variational circuit presented in Fig. 5.2 we also utilize the gate $R_y(\theta)$ which rotates a qubit around the Y-axis by the angle θ, given with:

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2). \end{pmatrix}$$

In Fig 5.2 we also denote the controlled-Z gate by the symbol:

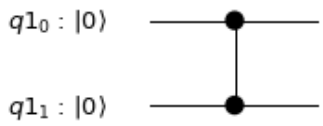


Figure 5.3: Controlled Z gate graphical representation

For completeness, we highlight that this gate (Cz) is represented with the following matrix:

$$C_Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

5.2.3 Circuits Generalization

Both circuits (feature mapping and variational) were designed/constructed to fixed number of qubits and depth. In order experiment with both 2-qubits and 3-qubits system, we need to generalize both circuits as such that it expands in dimensions based on number of qubits. For this, we followed the suggestions of [1], and developed our code as such that we first check the given f-depth of the circuit (in terms of the feature mapping circuit) and then apply the necessary gates mentioned previously. The circuit will act on $|0\rangle^n$ as initial state. The ϕ rule in this case is $\phi_i(x) = x_i$ and $\phi_{i,j}(x) = (\pi - x_i)(\pi - x_j)$. For the variational circuit, similar techniques were used to generalize the circuit. Note, we use only two-point ϕ functions, where a single angle depends at most on two entries of the input data vector. This can in principle be generalized. Finally, we again check the v-depth and n-qubits to apply the necessary gates.

5.3 Results - Artificial Data

5.3.0.1 Number of qubits = 2, Depth = 4

As a warm up to our study, we tested the performance of the quantum classifier on artificial datasets, explained in section 4.3. For all experiments here, we have fixed the shot numbers and the v-depth, but have explored the performance relative to the number of epochs (optimization steps), and total qubit numbers used.
Figure 5.4 tell us we almost have a near-perfect classification with a separation gap of magnitude 0.3 between them (white areas). In total we have 20 points per label training. The green data points, which are correctly classified, are on the green area and same

applies for the purple colored data points.

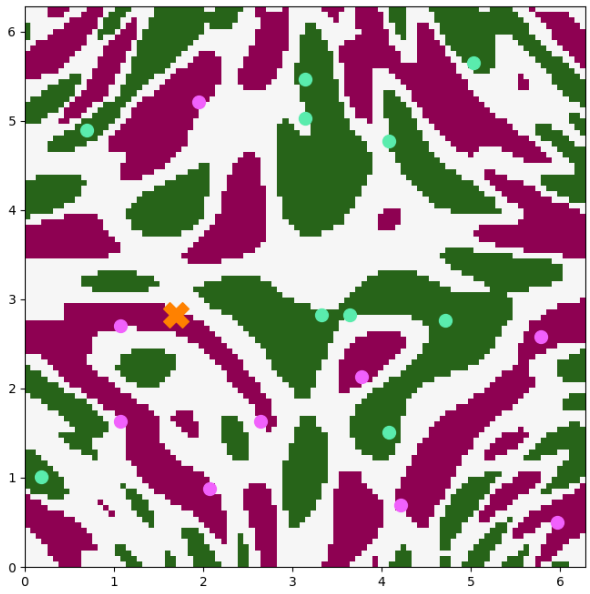


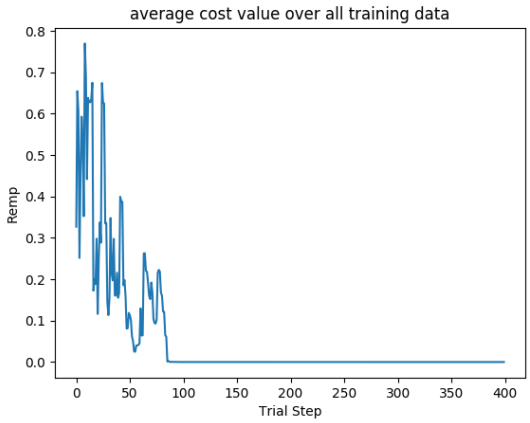
Figure 5.4: 20 data points separated by a magnitude of Δ=0.3 between them (white areas).

In table 5.5 we have our results for 2-qubit operation only. Here, we see that we already have a perfect classification of 100% using 200 epochs only. Increasing the epochs to 400 does not return the same results. In fact the results, in terms of classification on training and testing, are slightly decreased. But again, setting the epochs to 600 yields a fully classified data. Further investigation in terms of optimization steps simply does not guarantee the improvement of the training accuracy (or the test accuracy). Note that number of shots is a fixed number. In figures (5.5) we see the convergence of the cost function defined in Eq. 4.6 after 200 and 400 iterations/epochs. Here, we can see that the algorithm successfully learned the right hyperspace after 100-150 epochs and converged (almost) all the data points in the dataset. We will discuss the possible reasons for this instability shortly. Here, we are reproducing the results from the first method (quantum variational classifier) mentioned in [1], with similar settings and hyperparameters. Our

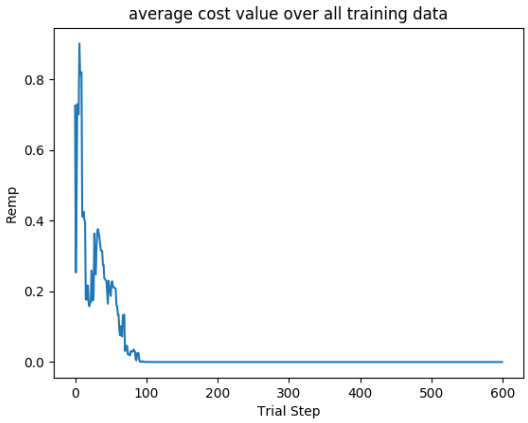
re-constructed model yields the same results as in [1]. Note, the results for artificial data is not cross-validated.

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	200	2000	100%	100%
2	4	400	2000	95%	90%
2	4	600	2000	100%	100%

Table 5.1: results of artificial data on 2-qubits



ACC training 95%



ACC training 100%

Figure 5.5: Average cost value over all training data where $n = 2$. Reminder: R_{emp} is the empirical risk (cost function) defined in 4.6 to reduce the error on the training set.

5.3.0.2 Number of qubits = 3, Depth = 4

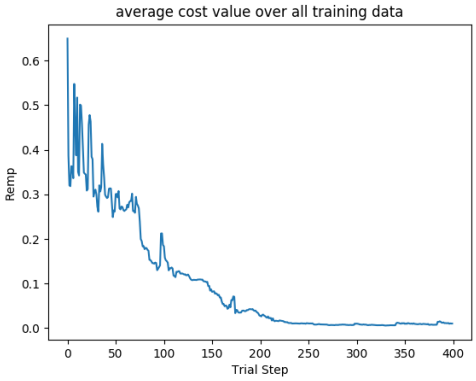
For 3-qubit system, another dimension z is added to the artificial data[1]. The ϕ rule in this case is $\phi_i(x) = x_i$ and $\phi_{i,j,z}(x) = (\pi - x_i)(\pi - x_j)(\pi - x_z)$.
We start with 200 epochs which yields 92% on training and 88% on testing. Increasing the optimization steps yields a good accuracy on both sets. However, as mentioned before, increasing the optimization steps does not guarantee good results hence the optimization steps we set to 800 which on the first run returned good results but on the second run

[1]Disclaimer: The codebase for artificial data was published by Qiskit team.

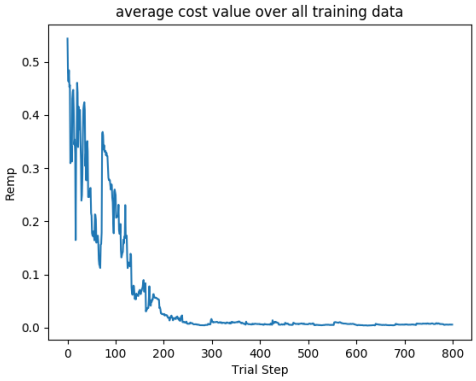
slightly decreased. In Figures 5.6 we can see how the cost function stabilizes the model, on each run (shot).

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	200	2000	92%	88%
3	4	400	2000	92%	90%
3	4	800	2000	95%	90%
3	4	800	2000	91%	88%

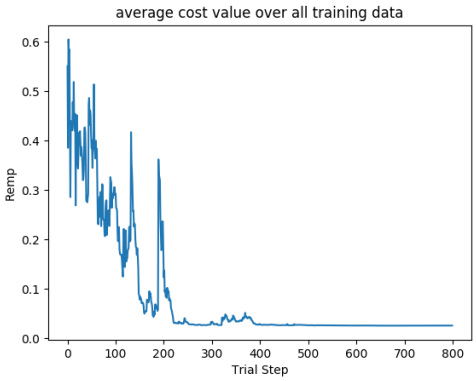
Table 5.2: results of artificial data on 3-qubits



ACC training 92%



ACC training 92%



ACC training 95%

Figure 5.6: Average cost value over all training data where $n = 3$.

5.3.0.3 Preliminary Discussion - Artificial Data

The results we obtain for n=2 and n=3 are consistent with the results reported in [1], so we are confident in our implementation. In both cases we see that it is likely that the optimization/training step is non-trivial. Already for n=2 the increase of the epochs (training steps) does not guarantee the improvement of the training accuracy (or the test accuracy). This effect is just more pronounced for n=3, where the search space is significantly larger. One aspect of the problem is that the optimizer may get stuck in a local minimum. Note that although the feature space is 4 and 8 dimensional for n=2,3 qubits, respectively, the search space of the optimizer is not exponential in the qubit number. The number of parameters are 40 and 70, for n=2 and n=3 respectively, which although are not scaling exponentially (like the feature space dimension), are still much larger than the feature space dimension for small n. This in principle may cause some overfitting for small n. Still it should not be the case that the optimizer gets stuck for this classification method to be useful. We will discuss this again later on. It is more likely that the convergence problems and instability come from the fact that not just the optimization method, but also the classifier itself is stochastic. That means that the performance evaluation may not be stable. We have confirmed this by running the 800 epoch training sequence twice, only to obtain significantly different results.

This highlights further need to investigate training procedures to achieve stable performance. However, due to the time limitations of this project, we leave these deeper studies for future work, and move on to the analysis of real-world datasets.

5.4 Results - Practical Data

The new quantum models provide a number of possible hyperparameters whose study is interesting. In this thesis we focus on perhaps the most fundamental parameter: number of qubits required. This is practically very important as large quantum computers are difficult to build. However, it also offers a conceptual challenge as we wanted to study the classifier performance on the *same* set for differing n, yet, this directly changes the

required dimension of the dataset. To this end we used PCA to to reduce the dimensionality of the dataset to match the dimensionality of the qubits we have. Consequently, the entire study does not just study the capacities of differing n in isolation, but coupled to a fixed dimensionality reduction technique. In practice, this is meaningful as the main proposals on how quantum classifiers should be used is always coupled with dimensionality reduction methods, as the quantum classifiers are limited in dimensions they can natively accept. Using PCA may lead to interesting effects, since for datasets where low PCA dimensions (n=2) suffice, using n=3 may lead to over-fitting, as the third principal component will not carry useful information. We will further discuss this in subsequent sections.

The following datasets were used for the experiments: Wine, BreastCancer and MNIST [48] which have 13, 30 and 64 dimensions, respectively. For all datasets we used 2 labels only. Operatively, this means we trained and tested our methods on subsets of the datasets. In terms of hyperparameters, we fixed de the depth of the variational circuit (v-depth) to 4 and feature mapping depth (f-depth) to 2. For each dataset, we tested our model on n=2 and n=3 qubits system. Number of shots (s) was fixed[1]. Furthermore, for a better model performance estimation we used cross-validation discussed in 3.3. For binary classification, the accuracy for evaluating classification models is measured. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{5.1}$$

[1]We tested various numbers starting from 200 to 1024 and in the end decided to use 2000. This, as mentioned earlier, the overall process can suffer from stochastic instabilities, and setting to shot number relatively high at least removed one source of stochastic noise. Also, we are adopting the same number of shots (for training/classification) mentioned in [1]

5.4.1 Wine dataset

Wine dataset [48] are related to red and white vinho verde wine samples, from the north of Portugal. The wine dataset has 13 attributes in total i.e., the dimension is 13, 3 classes, and number of instances are 178. One label has 80 instances and the other has 98.

5.4.1.1 2-qubits results

On 2-qubits, the results in table 5.3 are comparable. Note that the only change in hyper-parameters here is number of epochs, which we increase this by 200 for every run of the model. On this note, we started with 400 epochs that delivered us 77% accuracy on test set and 94% accuracy on training. With 600 epochs we are getting better results on both sets. However, increasing the epochs to 800 or 1000 does not seem to improve the results (i.e, the learning stops).

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	400	2000	94%	77%
2	4	600	2000	96%	84%
2	4	800	2000	95%	80%
2	4	1000	2000	95%	83%

Table 5.3: results of Wine dataset on 2-qubits

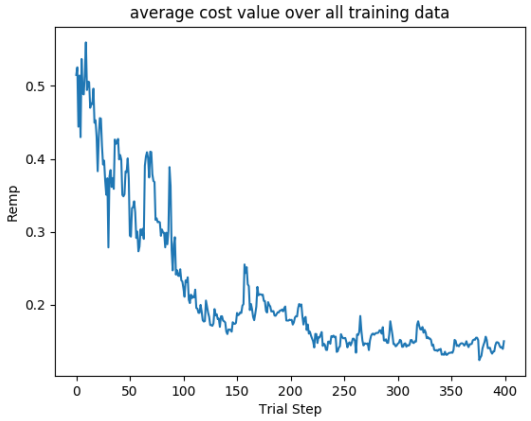
5.4.1.2 3-qubits results

In table 5.4, we increase number of qubits[1] to 3. Here, the results mostly suggest overfitting. The results are over-fitted because the model is actually low dimensional, and PCA delivers noise and this coupled with the fact that the model is actually quite powerful, it makes us learn noise. We will discuss this later. We also noticed that the accuracy on training did not yield higher results relative to n=2. This is indicative of the search space being big, and things being difficult to optimize. Also, the features may become complicated as there is a lot of effective noise in the third dimension.

[1]Note: Hilbert space grows rapidly with the size of a quantum system

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	400	2000	89%	60%
3	4	600	2000	88%	55%
3	4	800	2000	91%	64%
3	4	1000	2000	91%	64%

Table 5.4: results of Wine dataset on 3-qubits



ACC training 89%

ACC training 88%

ACC training 91%

ACC training 91%

5.4.2 Breast cancer dataset

UCI ML Breast Cancer[1] Dataset [48] contains 30 features and 569 samples. It has 2 labels, whether a cell mass is malignant or benign, and the goal of our classification system is to predict this. We split the data 114 as the testing size, and 454 for training.

5.4.2.1 2-qubits results

In table 5.5 we see that the generalization error as measured by performance on the testing set is quite low.. Again, starting with 400 epochs our classifier yields 97% accuracy on the training set and 88% on the test set. Increasing number of epochs to 600 yields the same accuracy on training and slightly improved accuracy on the test set. With 800 epochs, we see that the accuracy on the test set is further improved to 91% and 99% on the training set. In short, the results are suggestive of over-fitted, but the difference, in accuracy, between the testing and training sets sets is not big.

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	400	2000	97%	88%
2	4	600	2000	97%	89%
2	4	800	2000	99%	91%

Table 5.5: results of breast cancer dataset on 2-qubits

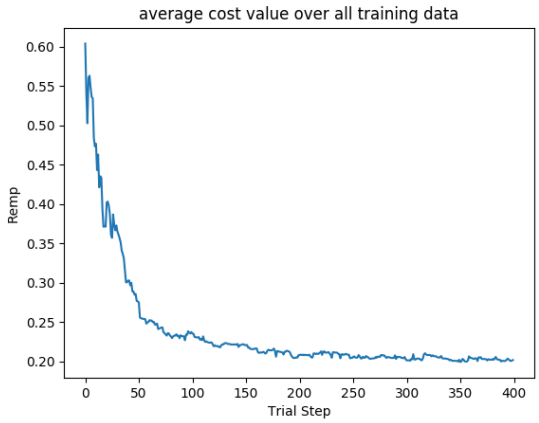
5.4.2.2 3-qubits results

On 3-qubits, the generalization error is higher than when using a 2-qubit system, which is indicative of overfitting. Here, the accuracy on training set is quite similar as on 2-qubits, but on testing set the results are drastically lower. There is little improvement between running the model on 400 epochs, which yields 71% accuracy on testing and 92% on training, and 600 epochs, which 73% on testing and 93% on training. Unfortunately, due to technical issues our run on 800 epochs was terminated before it was finished.

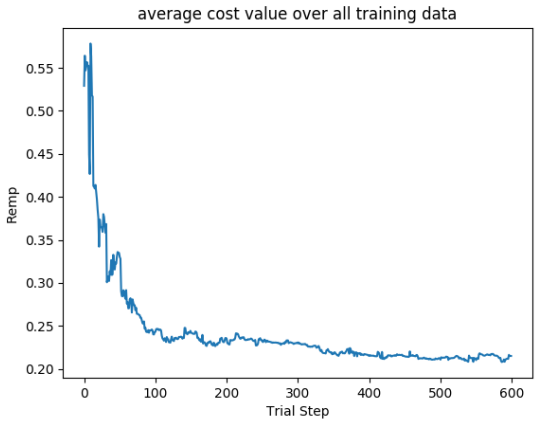
[1]Note: There are different sets of Breast Cancer. The one is we used is *diagnostic*, specifically made for classification task(s).

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	400	2000	92%	71%
3	4	600	2000	93%	73%

Table 5.6: results of Cancer dataset on 3-qubits



ACC training 92%



ACC training 93%

5.4.3 Digits dataset

MNIST dataset which has 10 classes, 1797 samples of handwritten grayscale digit, which are 8x8 images encoded as an unsigned integer (0-255). Finally, it has 64 dimensions. For this dataset we only considered 2 labels which are the subset of digits “0” and “1”. In the end we split the data 1437 as training set and 360 for testing.

5.4.3.1 2-qubits results

The results on MNIST dataset on 2-qubits is surprisingly good. Our tests on 400 epochs yields 89% accuracy on the testing set and 96% on training. However, on 600 and 800 epochs the models accuracy on both sets are slightly decreased.

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	400	2000	96%	89%
2	4	600	2000	96%	84%
2	4	800	2000	91%	87%

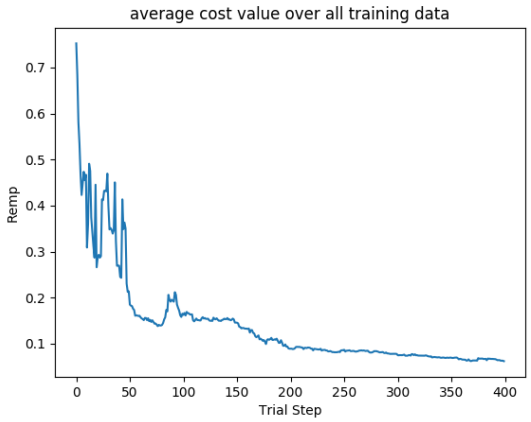
Table 5.7: results of MNIST dataset on 2-qubits

5.4.3.2 3-qubits results

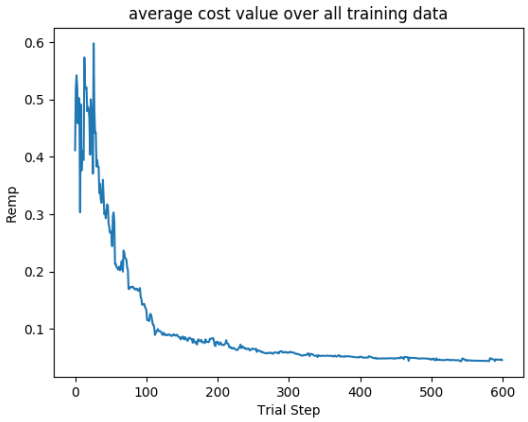
In the case of this third dataset, increasing the qubit number from n=2 to n=3 actually significantly improved performance. Running the model on 400 epochs yields a good 96% accuracy on the training and 88% on the testing set. Increasing epochs to 600 delivers better results, which are 97% on the training set and 90% on the testing set. In short, the results on both 2 or 3 qubits are suggestive of a slight overfitting, but overall we consider this a good fit.

qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	400	2000	96%	88%
3	4	600	2000	97%	90%
3	4	800	2000	97%	89%

Table 5.8: results of MNIST dataset on 3-qubits



ACC training 96%



ACC training 97%

5.5 Discussion

In this section we discuss how the experimental results elucidate the central questions of this thesis. As a warm-up, we first discuss our subsidiary question.

Subsidiary: *How does number of qubits affect the required number of optimization steps?*

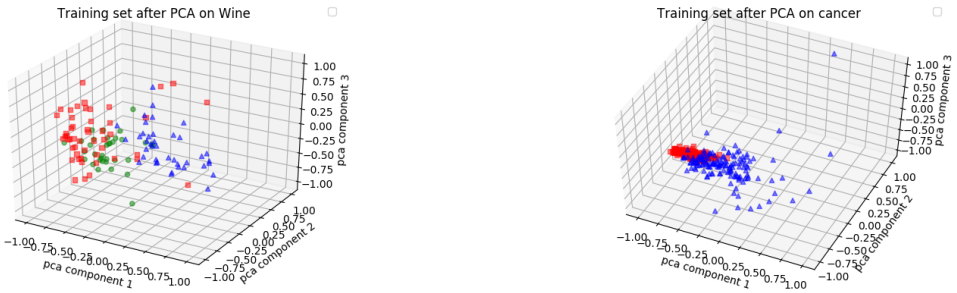
To monitor how the increase in the number of optimization steps affects the training and testing performance of the classifier, we trained our model many times and for each run we gradually increased number of optimization steps.
On 2-qubits system, and as mentioned before, with 200 epochs we already have 100% correctly classified artificial dataset. Here, our results are similar to the results mentioned in [1]. However, we see that the training may be unstable, as increasing the training times does not monotonically lead to better performance. Thus a study of how much training is actually optimal (or whether other optimizers are needed) is worth considering for follow-up work. Second, a separate study to see how much stochastic effects play a part is also of interest. This will be even more relevant when noisy and real QC implementations are considered.

Since the feature space dimension grows exponentially with n, n=3 already has a significantly larger search space. We suspect a main training issue is the stochasticity of the model, which makes training difficult. In most of our experiments, our classier is not able to classify the data points perfectly regardless number of shots or optimization steps. This can be caused either by having insufficient optimization steps (we cannot conclude this based on the trends we see, as increasing of the epoch numbers did not necessarily improve performance), or the issue may be other hyperparameters. Specifically, the depth of the circuit l is certainly a limitation of the expressivity of the model, and as n grows, l should actually grow exponentially to guarantee full expressivity for arbitrary datasets. This is a consequence of the fact that the size of the unitary operations on n qubits is exponential in n. Thus for low-depth circuits, it remains to be further investigated which datasets can be handled well. To study just the convergence properties, the artificial data

sets need to be tailored to be handled by the low-depth circuit. For real-world datasets, extensive experimentation is still warranted. This further motivates our main question: is using larger n beneficial in general at all.

> **Main question**: *How does using a larger shallow quantum circuits improve the classification performance on real datasets?*

This question is chosen as our main question as it is possibly most interesting. In the domain of shallow circuits (which roughly match the types of quantum computers we expect to have available in the near term), the "quantumness" of the system is not increased by deeper circuits, but just by their size (qubit number n). Thus it is an obviously interesting question how this hyperparameter influences the performance. However, as it turns out, this question is quite non-trivial, as the size n is, at least in the approach proposed in [1] which we adhered to, intertwined with the dimensionality of the data points the model takes. Thus to compare the performance of this hyperparameter *on the same dataset*, we must somehow manipulate the dimension(s). As mentioned before we used PCA to reduce the d-dimensional dataset to match the number of n qubits we have. A consequence of this is that we study not the pure effect of n on performance, but rather the performance is coupled with the fact that we use PCA and will thus also be sensitive to the type of the dataset we use, and whether PCA does useful feature extraction on it.



(a) Training set of Wine dataset after PCA (b) Training set of Cancer dataset after PCA

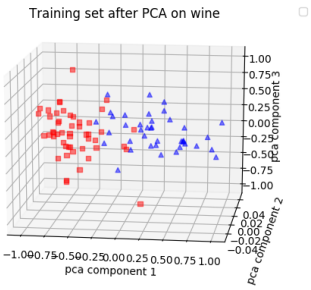
Figure 5.7: An example of PCA on the partitions after training set is split for the purposes of cross-validation.

Thus, strictly speaking, we study the the influence of the quantum SVM parameter n, together with PCA. Due to practical limitations of small sized quantum computers and our simulations, we will not be able to use large n-values as it is important to study the relationship between real data dimensions/complexity, a chosen dimensionality reduction technique, and the quantum SVM parameter n. This is not trivial as dimensionality reduction techniques themselves may make classification very simple, or unnecessarily complicated.

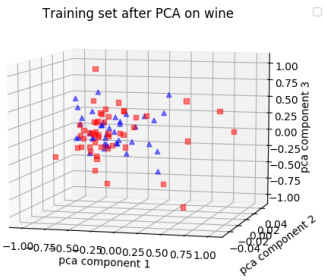
We have studied the effects the settings $n = 2$ and $n = 3$ on three datasets: Wine, Breastcancer and MNIST, with native dimensions 13, 30, 64, respectively. In general, based on Wine and Breastcancer, we cannot conclude that increasing n improves performance. It is worsened, both for testing and training performance. In contrast, MNIST dataset performance does seem to improve, both in terms of training and generalization performance. We have a few possible explanations for this phenomenon, all of which can be traced to the use of PCA. The worsening of the training performance on the lower-dimensional datasets suggest that the PCA creates an increasingly difficult to separate/classify landscape. Our assumption is that the third dimension of the PCA is essentially capturing noise (See Fig. 5.8) as the correct labeling is essentially fully captured by the first two principal components.

In this case, our model may either fail to even train properly, and even if it did, it would most likely have poor generalization performance as noise is learned. In the case of MNIST sets, it is known that higher principal components still do carry information about the labels, hence the performance is not decreased, but, as expected improved as n increased. To test some of our conjectures, we explored the structure of the PCA outputs where n =3 (see Fig 5.7).

(a) PCA applied on Wine training set - two components X and Y axis - Side view



(b) PCA applied on Wine training set - two components X and Z axis - Side View



(c) PCA applied on Wine training set - two components Y and Z axis - Side view

Figure 5.8: An example of PCA applied to Wine training set. Here, we can point out that axis XY and XZ have valuable information/components. However, the axis YZ do not correctly group the labels, so learning it would lead to overfitting.

Beyond the issues involving PCA, even on the bare level of the quantum classifier, it is not obvious to what extent the increasing n increases the "richness" of the classifying functions. Intuitively, by increasing the number of qubits, thereby increasing the dimensionality of the feature space, we should be able to better separate/classify complex landscapes. However, recall that since we work with fixed-depth variational circuits, the power/ability of our variational circuit to exploit the high-dimensionality of the feature space decreases (in a proportional sense) when we increase the number of qubits. Specifically, in the SVM interpretation of the quantum SVM algorithm the purpose of the variational circuit is to rotate the fixed hyperplane, specified by the measurement, into

the optimal position for classification. However, since our circuit is shallow, we cannot implement all (unitary) rotations, and, roughly speaking, the "density" of all unitaries we can implement with fixed depth decreases as the dimension grows - even though the total parameter number grows. The only example where this effect may have been observed is the MNIST dataset, but to study this in more detail, we would require to go to higher dimensions beyond n=3, which was beyond our computational capacities. This remains an interesting question for a follow up work.

Chapter 6

Conclusions

Quantum machine learning aims to harness the laws and properties of quantum mechanics to outperform its classical counterparts. It is expected that quantum computers will play a significant role in many data processing fields, and also specifically in the domain of big data, where the datasets are pushing our conventional computational resources to their limits.
This thesis investigated the potential of one class of quantum classifiers for supervised learning. We have build upon the approaches of [1], and have implemented the quantum classifier which exploits the exponentially large quantum features of the quantum Hilbert space. Here our goal was to investigate new types of model parameters, such as the qubit number and optimization steps and how this influence the classification performance.

To summarise our approach, we utilized a feature map to encode (or transform) data $\vec{x} \in \mathbb{R}^n$ according to the equation defined in Eq. 4.2, depending on the depth, we duplicate the same circuit d times, where d is the depth of the circuit. Secondly, a short depth quantum circuit defined in 4.5 is applied to the feature state. This circuit with l-layers is parameterized by $\theta \in \mathbb{R}^{2n(l+1)}$ that was optimized during the training. Afterwards, label classification and measurement takes place as described in 4.2.2. Based on these results and experiments we conducted, we can conclude that our quantum classifier can learn to a certain degree (i.e., it has good convergence rate on average on all datasets) where it returns a good accuracy on both the training and testing sets. Also, we have found out that even by reducing dimensionaity of the data via PCA, our classifier is

still capable of achieving relatively good accuracy on both 2 and 3 qubits. Note that our tests and experiments were executed on a simulated qubit(s) using Qiskit [27]. Due to technical limitations with IBM Quantum Experience, we were not able to run our experiments on a real quantum-chip. In this work we have opted to use PCA as the dimensionality technique to map the data onto the limited quantum classifier. However, intuitively, the quantum classifiers strengths are likely to lie in its capacity to decode very complex data landscapes. Consequently, to make the most out of it, it makes sense to experiment with broader classes of dimensionality reduction techniques. For instance, using "*Uniform Manifold Approximation and Projection* (UMAP)" [50] maybe a good idea. In this thesis we focused on the performance of the idealized quantum classifier which was executed with no noise. As near-term quantum devices will certainly be quite noisy, it would be important to re-run similar experiments using noise simulators (such as Qiskit ignis package [51]) or using actual quantum computers. While the results of this thesis highlight many potential pitfalls of near-term quantum classifiers, it is obvious that the potential of these upcoming quantum technologies should not be overlooked.

References

[1] V. Havlíček, A. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, pp. 209–212, 03 2019. i, 2, 5, 6, 19, 22, 23, 24, 25, 26, 30, 35, 36, 37, 39, 40, 46, 47, 51

[2] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, "Stock market prediction system with modular neural networks," in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 1–6 vol.1, June 1990. 1

[3] H. Mizuno, M. Kosaka, H. Yajima, and N. Komoda, "Application of neural network to technical analysis of stock market prediction," vol. 7, 01 1998. 1

[4] M. A. Holmstrom and D. Z. Liu, "Machine learning applied to weather forecasting," 2016. 1

[5] G. K. Venayagamoorthy, V. Moonasar, and K. Sandrasegaran, "Voice recognition using neural networks," in *Proceedings of the 1998 South African Symposium on Communications and Signal Processing-COMSIG '98 (Cat. No. 98EX214)*, pp. 29–32, Sep. 1998. 1

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015. 1

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv*, vol. 1409, 09 2014. 1

[8] I. M Johnstone and D. Michael Titterington, "Statistical challenges of high-dimensional data," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 367, pp. 4237–53, 11 2009. 1, 3

[9] F. Lardinois, "Uk government $194m to commercialize quantum computing," 06 2019. 2

[10] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, 09 2014. 2, 4

[11] "Model Generalization: challenges of generalization in machine learning." `https://bit.ly/2TkuoX0`. Accessed: 2019-06-27. 3

[12] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016. 3

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 3

[14] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, 11 2016. 3

[15] V. Dunjko and H. J. Briegel, "Machine learning & artificial intelligence in the quantum domain," 09 2017. 3, 4

[16] J. Preskill, "Quantum computing and the entanglement frontier - rapporteur talk at the 25th solvay conference," 03 2012. 3

[17] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, pp. 1484–1509, 10 1997. 3

[18] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), pp. 212–219, ACM, 1996. 3

[19] E. Pednault, J. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff, "Google plans to demonstrate the supremacy of quantum computing," 10 2017. Accessed: 2019-06-17. 3

[20] E. Pednault, J. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff, "Breaking the 49-qubit barrier in the simulation of quantum circuits," 10 2017. 3

[21] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," 07 2013. 4

[22] P. Wittek, *Quantum Machine Learning: What Quantum Computing Means to Data Mining*. 08 2014. 4

[23] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Physical Review Letters*, vol. 113, 07 2013. 4, 5

[24] M. Benedetti, E. Lloyd, and S. Sack, "Parameterized quantum circuits as machine learning models," *arXiv e-prints*, p. arXiv:1906.07682, Jun 2019. 4

[25] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, p. 023023, feb 2016. 4, 25

[26] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. New York, NY, USA: Cambridge University Press, 10th ed., 2011. 10

[27] " Qiskit: An open-source quantum computing framework for leveraging today's quantum processors in research, education, and business)." `https://qiskit.org/`. Accessed: 2019-05-31. 12, 32, 34, 52

[28] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995. 14, 16, 17

[29] "Javier Ruiz Hidalgo: Methodology (dlai d6l2 2017 upc deep learning for artificial intelligence))." 15

[30] " Sebastian Raschka: Machine learning faq)." `https://sebastianraschka.com/faq/docs/evaluate-a-model.html`. Accessed: 2019-05-31. 16

[31] ritchieng, "Cross-validation explained," 09 2018. Accessed: 2019-07-15. 16

[32] "Support Vector Machine support vector machines." `https://www.learnopencv.com/support-vector-machines-svm/`. Accessed: 2019-05-13. 17

[33] "Support Vector Machine support vector machines." `https://bit.ly/2pFAPFo`. 17

[34] . P. F. . Schuld, M., "Supervised learning with quantum computers," *Springer*, vol. 98, 03 2018. 18

[35] "Cross entropy: cross-entropy is commonly used to quantify the difference between two probability distributions.." `https://en.wikipedia.org/wiki/Cross_entropy`. Accessed: 2019-05-13. 19

[36] "Curse of dimensionality curse of dimensionality." `https://en.wikipedia.org/wiki/Curse_of_dimensionality`. Accessed: 2019-05-13. 19

[37] S. Lloyd, M. Mohseni, and P. Rebentrost, "About the curse of dimensionality," 07 2018. Accessed: 2019-07-08. 20

[38] " PCA: Understanding principal component analysis." `https://bit.ly/2YCAckx`. 20

[39] " PCA: Principle component analysis)." `https://bit.ly/2Ma1Kaq`. Accessed: 2019-06-02. 20

[40] B. Skuse, "The trouble with quantum computing," 06 2019. 21

[41] J. C. Spall and S. Member, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, pp. 332–341, 1992. 21

[42] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, 03 2018. 22

[43] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," 02 2018. 22

[44] " Qiskit: The second order expansion feature map from qiskit." 23

[45] N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, and K. Temme, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, vol. 3, p. 030503, jun 2018. 25

[46] "Variational Circuits variational circuits (also called parametrized quantum circuits)." `https://qmlt.readthedocs.io/en/latest/variational.html`. Accessed: 2019-05-05. 27

[47] " Dawid Kopczyk: Enthusiastically about algorithms)." `https://bit.ly/2YVTbSo`. Accessed: 2019-06-31. 28

[48] D. Dua and C. Graff, "UCI machine learning repository," 2017. 31, 40, 41, 43

[49] "SPSA: simultaneous perturbation stochastic approximation." `https://qiskit.org/documentation/aqua/optimizers.html`. Accessed: 2019-05-13. 32

[50] L. McInnes, J. Healy, N. Saul, and L. Grossberger, "Umap: Uniform manifold approximation and projection," *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018. 52

[51] " Qiskit Ignis: is a framework for understanding and mitigating noise in quantum circuits and devices.)." `https://qiskit.org/ignis`. Accessed: 2019-05-08. 52

[52] " Qiskit Terra: provides the foundational roots for our software stack.)." `https://qiskit.org/terra`. Accessed: 2019-05-08.

[53] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?," *Physical Review*, vol. 47, pp. 777–780, May 1935.

[54] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[55] L. Lamata, "Basic protocols in quantum reinforcement learning with superconducting circuits," *Scientific Reports*, vol. 7, p. 1609, 05 2017.

[56] C. Zoufal, A. Lucchi, and S. Wörner, "Quantum generative adversarial networks for learning and loading random distributions," 03 2019.

[57] E. Aimeur, G. Brassard, and S. Gambs, "Quantum speed-up for unsupervised learning," *Machine Learning*, vol. 90, pp. 261–287, 02 2013.

[58] V. Dunjko, J. Taylor, and H. J. Briegel, "Quantum-enhanced machine learning," *Physical Review Letters*, vol. 117, 09 2016.

[59] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum principal component analysis," *Nature Physics*, vol. 10, 07 2013.