Mark Fingerhuth

DRAFT:
Quantum-enhanced machine learning: Implementing a quantum k-nearest neighbour algorithm

Bachelor Thesis

In partial fulfillment of the requirements for the degree Bachelor of Science (BSc) at Maastricht University.

Supervision

Prof. Francesco Petruccione
Dr. Fabrice Birembaut

January 2017

Preface

This thesis is submitted for the Bachelor degree at the University of Maastricht. The research for this thesis was conducted under the supervision of Prof. F. Petruccione at the Centre for Quantum Technology, University of Kwaluzu-Natal, South Africa between September 2016 and January 2017.

This work is to the best of my knowledge original, expect where acknowledgments and references are made to previous work. Neither this, nor any substantially similar thesis has been or is being submitted for any other degree, diploma or other qualification at any other university.

No part of this thesis has been previously published.

<div align="right">

Mark Fingerhuth

January 2017

</div>

Acknowledgements

Contents

Abstract

NEEDS MODIFICATION!

Quantum machine learning, the intersection of quantum computation and classical machine learning, bears the potential to provide more efficient ways to deal with big data through the use of quantum superpositions, entanglement and the resulting quantum parallelism. The proposed research will attempt to implement and simulate two quantum machine learning routines and use them to solve small machine learning problems. That will establish one of the earliest proof-of-concept studies in the field and demonstrate that quantum machine learning is already implementable on small-scale quantum computers. This is vital to show that an extrapolation to larger quantum computing devices will indeed lead to vast speed-ups of current machine learning algorithms.

Nomenclature

Symbols

$\otimes$	Tensor product	
i	Imaginary unit	$i = \sqrt{-1}$
$\dagger$	Hermitian conjugate	Complex conjugate transpose
!	Factorial	e.g. 3! = 3*2*1

Acronyms and Abbreviations

CCNOT	Controlled controlled NOT gate (Toffoli gate)
CM	Conditional measurement
CNOT	Controlled NOT gate
CU	Controlled U gate (where U can be any unitary quantum gate)
GB	Gigabyte
HD	Hamming distance
ML	Machine learning
PM	Post-measurement
Prob	Probability
QC	Quantum computer
QML	Quantum machine learning
RAM	Random-access memory

Chapter 1

Introduction

The ability to understand spoken language, to recognise faces and to distinguish types of fruit comes naturally to humans, even though these processes of pattern recognition and classification are inherently complex. Machine learning (ML), a subtopic of artificial intelligence, is concerned with the development of algorithms that perform these types of tasks, thus enabling computers to find and recognise patterns in data and classify unknown inputs based on previous training with labelled inputs. Such algorithms make the core of e.g. human speech recognition and recommendation engines as used by Amazon.

According to IBM (2016b), approximately 2.5 quintillions (10^{18}) bytes of digital data are created every day. This growing number implies that every area dealing with data will eventually require advanced algorithms that can make sense of data content, retrieve patterns and reveal correlations. However, most ML algorithms involve the execution of computationally expensive operations and doing so on large data sets inevitably takes a lot of time (Bekkerman, Bilenko, & Langford, 2011). Thus, it becomes increasingly important to find efficient ways of dealing with big data.

A promising solution is the use of quantum computation which has been researched intensively in the last few decades. Quantum computers (QCs) use quantum mechanical systems and their unique properties to manipulate and process information in ways that are impossible to implement on classical computers. The quantum equivalent of a classical bit is called a quantum bit (qubit) and additionally to being in either state $|0\rangle$ or $|1\rangle$ it can be in their linear superposition.

This peculiar property gives rise to so-called quantum parallelism, which enables the execution of certain operations on many quantum states at the same time. Despite this advantage, the difficulty in quantum computation lies in the retrieval of the computed solution since a measurement of a qubit collapses it into a single classical bit and thereby destroys information about its previous superposition. Several quantum algorithms have been proposed that provide exponential speed-ups when compared to their classical counterparts with Shor's prime factorization algorithm being the most famous (Shor, 1994). Hence, quantum computation has the potential to vastly improve computational power, speed up the processing of big data and solve certain problems that are practically unsolvable on classical computers.

Considering these advantages, the combination of quantum computation and classical ML into the new field of quantum machine learning (QML) seems almost natural. There are currently two main ideas on how to merge quantum computation with ML, namely a) running the classical algorithm on a classical computer and outsourcing only the computationally intensive task to a QC or b) executing the quantum version of the entire algorithm on a QC. Current QML research mostly focusses on the latter approach by developing quantum algorithms that tap into the full potential of quantum parallelism.

1.1   Motivation

Classical ML is a very practical topic since it can be directly tested, verified and implemented on any commercial classical computer. So far, QML has been of almost entirely theoretical nature since the required computational resources are not in place yet. To yield reliable solutions QML algorithms often require a relatively large number of error-corrected qubits and some quantum data storage such as the proposed quantum random access memory (qRAM) (Giovannetti, Lloyd, & Maccone, 2008). However, to date the maximum number of superconducting qubits reportedly used for calculation is nine, the D-Wave II quantum annealing device delivers 1152 qubits but can only solve a narrow class of problems, and a qRAM has not been developed yet (O'Malley et al., 2016; D-Wave, 2015). Furthermore, qubit error-correction is still a very active research field and most of the described preliminary QCs deal with non-error-corrected qubits with short lifetimes and are, thus, impractical for large QML implementations.

Until now there have been only a few experimental verifications of QML algorithms that establish proof-of-concept. Li, Liu, Xu, and Du (2015) successfully distinguished a handwritten six from a nine using a quantum support vector machine on a four-qubit nuclear magnetic resonance test bench. In addition, Cai et al. (2015) were first to experimentally demonstrate quantum machine learning on a photonic QC and showed that the distance between two vectors and their inner product can indeed be computed quantum mechanically. Lastly, Ristè et al. (2015) solved a learning parity problem with five superconducting qubits and found that a quantum advantage can already be observed in non-error-corrected systems.

Considering the gap between the number of proposed QML algorithms and the few experimental realisations, it remains important to find QML problems which can already be implemented on small-scale QCs. Hence, the purpose of this study is to provide proof-of-concept implementation of selected QML algorithms on small datasets. This is an important step in the attempt to shift QML from a purely theoretical research area to a more applied field such as classical ML.

1.2   Research question

In light of the theoretical nature of current QML research and the small number of experimental realisations, this research will address the following question:

How can a k-nearest neighbour quantum machine learning algorithm be implemented on small-scale quantum computers?

The following sections will outline the steps required, and the tools used to answer this research question.

Chapter 2

Theoretical Foundations

2.1 Quantum bits

2.1.1 Single qubit systems

Classical computers manipulate bits, whereas quantum computer's most fundamental unit is called a quantum bit, often abbreviated as qubit. Bits, as well as qubits, are binary entities, meaning they can only take the values 0 and 1.

A good example of a physical implementation of a qubit is the single electron of a hydrogen atom sketched in Fig. 2.1. Usually, the electron is found in its ground state which one can define as the $|0\rangle$ state of the qubit. By using a laser pulse, the electron can be excited into the next highest valence shell which one can define as the $|1\rangle$ state of the qubit. After some time t the electron will decohere to its ground state ($|0\rangle$) which is called the longitudinal coherence or amplitude damping time and is an important parameter for measuring qubit lifetimes.
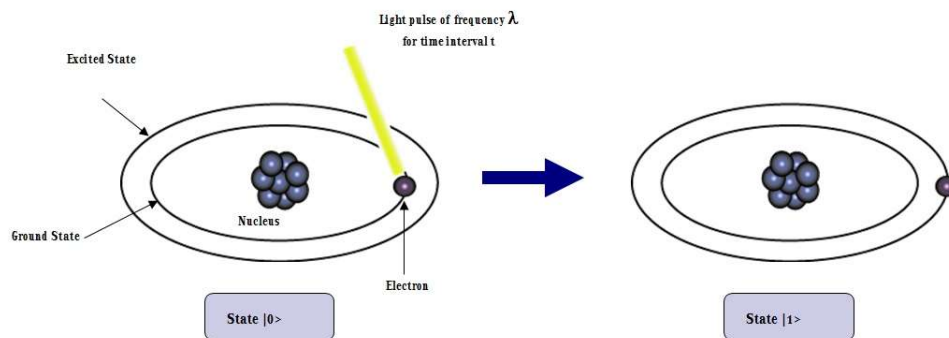

Figure 2.1: A simple example of a physical qubit using the electron of a hydrogen atom[1]

A classical non-probabilistic bit can only be in one of the two possible states at once. In contrast, qubits obey the laws of quantum mechanics, which gives rise to the important property that - besides being a definite 0 or 1 - they can also be in a superposition of the two states. Mathematically this is expressed as a linear combination of the states 0 and 1:

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

where α and β are complex coefficients ($\alpha, \beta \in \mathbb{C}$) and are often referred to as amplitudes. Any amplitude η can be further subdivided into a complex phase factor $e^{i\phi}$ and a non-negative real number e such that,

$$\eta = e^{i\phi} e \tag{2.2}$$

In Equ. 2.1, $|0\rangle$ is the Dirac notation for the qubit being in state 0 and it represents a two-dimensional vector in a complex 2-D vector space (called Hilbert space $\mathcal{H}_2$). $|0\rangle$ and $|1\rangle$ are the computational basis states and they constitute an orthonormal basis of $\mathcal{H}_2$. For the sake of clarity, $|0\rangle$ and $|1\rangle$ can be thought of as the 2-D vectors shown below.

APPROXIMATELY EQUAL SIGN!

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.3}$$

Subbing these vectors into Equ. 2.1 yields the vector representation of $|\psi\rangle$:

$$|\psi\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2.4}$$

In the cases $\alpha = 1$ or $\beta = 1$ there is no quantum behaviour since the qubit is not in a superposition. However, if for example $\alpha = \beta = \frac{1}{\sqrt{2}}$ the qubit is in an equal quantum superposition which is impossible to achieve with a classical computer. This leads to another important qubit lifetime parameter - the transversal coherence or phase damping time. It is measured by preparing the equal superposition $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and due to unavoidable interaction with the environment after some time t the quantum behaviour will be lost, and the state will either be a definite $|0\rangle$ or $|1\rangle$. The process of losing quantum behaviour is called *decoherence*.

However, even though a qubit can be in a superposition of $|0\rangle$ and $|1\rangle$, when measured it will take the value $|0\rangle$ with a probability of

$$Prob(|0\rangle) = |\alpha|^2 \tag{2.5}$$

and $|1\rangle$ with a probability of

$$Prob(|1\rangle) = |\beta|^2 \tag{2.6}$$

The fact that the probability of measuring a particular state is equal the absolute value squared of the respective amplitude is called Born rule (citation). Since the total probability of measuring any value has to be 1, the following normalization condition must be satisfied:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.7}$$

Therefore, a qubit is inherently probabilistic but when measured it collapses into a single classical bit (0 or 1). It follows that a measurement destroys information about the superposition of the qubit (the values of α and β). This constitutes one of the main difficulties when designing quantum algorithms since only limited information can be obtained about the final states of the qubits in the quantum computer.

Using spherical polar coordinates, a single qubit can be visualized on the so-called Bloch sphere by parameterising α and β in Equ. 2.1 as follows:

$$|q\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.8}$$

The Bloch sphere has a radius of 1 and is, therefore, a unit sphere. The $|0\rangle$ qubit state is defined to lie along the positive z-axis ($\hat{z}$) and the $|1\rangle$ state is defined to lie along the negative z-axis ($-\hat{z}$) as labelled in Fig. 2.2. At this point, it is important to note that these two states are mutually orthogonal in $\mathcal{H}_2$ even though they are not orthogonal on the Bloch sphere.

Qubit states on the Bloch equator such as the $\hat{x}$ and $\hat{y}$ coordinate axes represent equal superpositions where $|0\rangle$ and $|1\rangle$ both have measurement probabilities equal to 0.5. The $\hat{x}$-axis for example represents the equal superposition $|q\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. As illustrated in Fig. 2.2 any arbitrary 2-D qubit state $|\psi\rangle$ can be decomposed into the polar angles θ and ϕ and visualized as a vector on the Bloch sphere. Such an object is called the Bloch vector of the qubit state $|\psi\rangle$. The Bloch sphere will be the main visualization tool for qubit manipulations in this thesis.
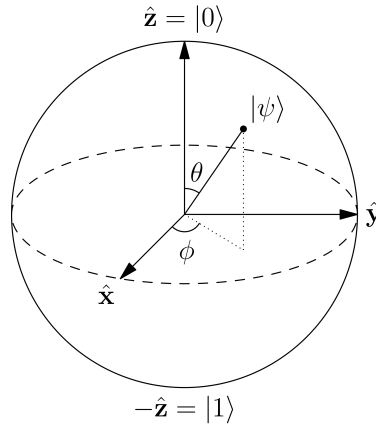
Figure 2.2: An arbitrary two-dimensional qubit $|\psi\rangle$ visualized on the Bloch sphere.[2]

Similar to logic gates in a classical computer, a QC manipulates qubit by means of quantum logic gates which will be introduced in detail in Section 2.2. Generally, an arbitrary quantum logic gate U acting on a single qubit state is a linear transformation given by a 2x2 matrix whose action on $|\psi\rangle$ is defined as:

$$U|\psi\rangle = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a*\alpha + b*\beta \\ c*\alpha + d*\beta \end{pmatrix} \tag{2.9}$$

Matrix U must be unitary which means that a) its determinant is equal to unity and b) its Hermitian conjugate $U^\dagger$ must be equal to its inverse (see Equ. 2.10 and 2.11). All quantum logic gates must be unitary since this preserves the normalisation of the qubit state it is acting on.

$$a)\ |\ det(U)\ |= 1 \tag{2.10}$$
$$b)\quad UU^\dagger = U^\dagger U = \mathbb{1} = UU^{-1} = U^{-1}U \tag{2.11}$$

The set of all 2-D complex unitary matrices with determinant one is called the special unitary group $SU(2)$ and all single-qubit quantum gates are therefore elements of $SU(2)$. Furthermore, a gate set G consisting of m quantum-gates $g_1, g_2, ..., g_m$ is called a *universal quantum gate set* when it is a dense subset of $SU(2)$ as defined in the red box below.

Definition: Dense Subset of $SU(2)$

The gate set G is a dense subset of $SU(2)$ when given any quantum gate $W \in SU(2)$ and any accuracy $\epsilon > 0$ there exists a product J of gates from G which is an ϵ-approximation to W (Dawson & Nielsen, 2005).

2.1.2 Multi qubit systems

A classical computer with one bit of memory is not particularly useful and equally, a QC with one qubit is rather useless. To be able to perform large and complicated computations many individual qubits need to be combined to create a large QC. When moving from single to multi-qubit systems a new mathematical tool, the so-called tensor product (symbol $\otimes$), is needed. A tensor product of two qubits is written as:

$$|\psi\rangle \otimes |\psi\rangle = |0\rangle \otimes |0\rangle = |00\rangle \tag{2.12}$$

Whereby the last expression omits the $\otimes$ symbol which is the shorthand form of a tensor product between two qubits. A *quantum register* of size j is an alternative way of referring to a tensor product of j qubits. In QML algorithms, large quantum states are usually subdivided into several quantum registers fulfilling different purposes e.g. storing data or class label.

In vector notation a tensor product of two vectors (red and green) is defined as shown below:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 * \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 0 * \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.13}$$

The last expression in Equ. 2.13 shows that the two-qubit state $|00\rangle$ is no longer two but four-dimensional. Hence, it lives in a four-dimensional Hilbert space $\mathcal{H}_4$. A quantum gate acting on multiple qubits can therefore not have the same dimensions as a single-qubit gate (Equ. 2.9) which demands a new gate formalism for multi-qubit systems.

Consider wanting to apply an arbitrary single-qubit gate U (Equ. 2.9) to the first qubit and leaving the second qubit unchanged, essentially applying the identity matrix $\mathbb{1}$ to it. To do this, one defines the tensor product of two matrices as follows,

$$U \otimes \mathbb{1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & b * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ c * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & d * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{pmatrix} \tag{2.14}$$

Thus, the result of the tensor product $U \otimes \mathbb{1}$ is a unitary 4x4 matrix which can now be used to linearly transform the 4x1 vector representing the $|00\rangle$ state in Equ. 2.13:

$$U \otimes \mathbb{1} |00\rangle = \begin{pmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ 0 \\ c \\ 0 \end{pmatrix} \tag{2.15}$$

One can also first perform the single qubit operations on the respective qubits followed by the tensor product of the two resulting vectors:

$$(U \otimes \mathbb{1})(|0\rangle \otimes |0\rangle) = U |0\rangle \otimes \mathbb{1} |0\rangle = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ c \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ 0 \\ c \\ 0 \end{pmatrix} \tag{2.16}$$

This formalism can be extended to any number of qubits and the use of tensor products leads to an exponential increase in the dimensionality of the Hilbert space. Hence, n qubits live in a 2^n-dimensional Hilbert space ($\mathcal{H}_2^{\otimes n}$) and can store the content of 2^n classical bits. As an example, only 33 qubits can store the equivalent of $2^{33} = 8,589,934,592$ bits ($= 1$ gigabyte) which clearly bears the potential for enormous speed-ups in computations as will be demonstrated later.

When considering multi qubit systems one will encounter quantum states that can or cannot be factorized. Consider for example the last expression in Equ. 2.16 which can be reexpressed as follows,

$$\begin{pmatrix} a \\ 0 \\ c \\ 0 \end{pmatrix} = a \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + c \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + 0 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = a |00\rangle + 0 |01\rangle + c |10\rangle + 0 |11\rangle \tag{2.17}$$

This can be factorized into the following tensor product:

$$a |00\rangle + c |10\rangle = (a |0\rangle + c |1\rangle) \otimes |0\rangle \tag{2.18}$$

In contrast, consider the following state:

$$|\phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.19}$$

It is straightforward to see that there is no way to factorise the state $|\phi\rangle$ into a tensor product of two states. Now imagine, two people Andy and Beatrice are given two electrons prepared in the quantum state $|\phi\rangle$. Andy keeps the first electron in the lab and Beatrice takes the second electron to her house. Andy gets interested in measuring if his electron is in the $|0\rangle$ or $|1\rangle$ state and performs a measurement. He finds the electron to be in the $|1\rangle$ state. From Equ. 2.19 and knowing that measurement collapses a superposition the post-measurement (PM) state $|\phi\rangle_{PM}$ is given by Equ. 2.20.

$$|\phi\rangle_{PM} = |11\rangle \tag{2.20}$$

Looking at this expression tells Andy that Beatrice's electron must be in state $|1\rangle$ as well without having measured her electron! Even more surprising is the fact that the second electron was nowhere close to Andy and he was still able to determine its state.

Non-local correlations between qubit measurement outcomes is a peculiar quantum property of non-factorising quantum states and is called *entanglement*. It is an integral part of quantum computation and Section 2.2.2 will give a concrete example of how to create such a state in a QC.

2.2 Quantum Logic Gates

In order to perform quantum computations, tools, analogous to the classical logic gates, are needed for qubit manipulation. Quantum logic gates are square matrices that can be visualised as rotations on the Bloch sphere. The following subsections will introduce the major single and multi-qubit logic gates.

2.2.1 Single qubit gates

Qubit flip (X) gate

The quantum equivalent of the classical NOT logic gate is called X gate and is given by the 1st Pauli matrix:

$$\sigma_1 = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.21}$$

The action of a quantum gate on the arbitrary qubit state $|\psi\rangle$ (Equ. 2.4) can be analysed using the gate's matrix and the qubit's vector representation. Applying some straightforward linear algebra yields:

$$X \otimes |\psi\rangle = X \otimes (\alpha |0\rangle + \beta |1\rangle) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} = \beta |0\rangle + \alpha |1\rangle \tag{2.22}$$

Thus, applying the X gate to qubit state $|\psi\rangle$ swaps the amplitudes of $|0\rangle$ and $|1\rangle$. More specifically, applying X to the $|0\rangle$ state results in the $|1\rangle$ state,

$$X \otimes |0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{2.23}$$

In terms of the example with the electron of a hydrogen atom shown in Fig. 2.1 an X gate can be implemented by exciting the electron from the ground state ($|0\rangle$) into the next highest electron valence shell ($|1\rangle$) using a controlled laser pulse.

The $|0\rangle$ state is recovered when applying X again to the $|1\rangle$ state,

$$X \otimes |1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{2.24}$$

On the Bloch sphere, the X gate corresponds to an anti-clockwise π rotation around the x-axis as shown in Fig. 2.3.
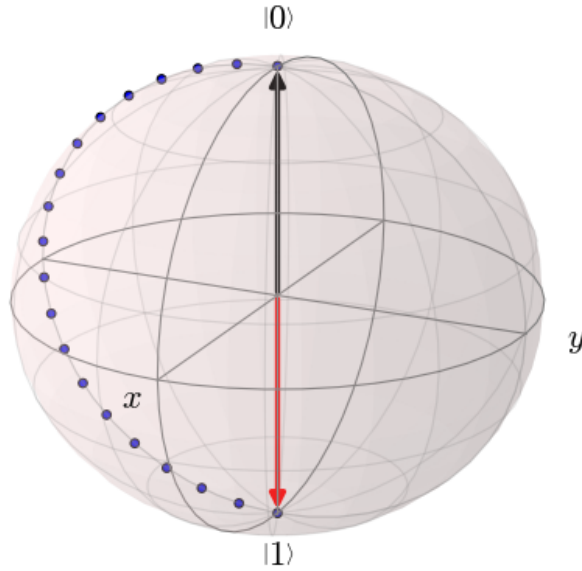
Figure 2.3: Visualization of the X gate on the Bloch sphere. Initial state (black) transformed to final state (red).
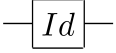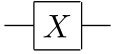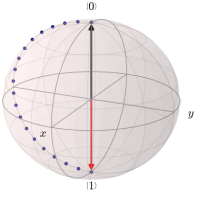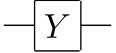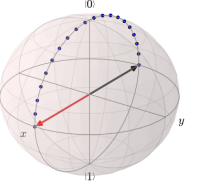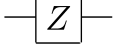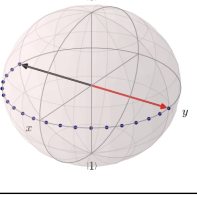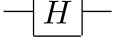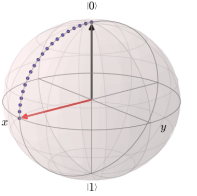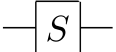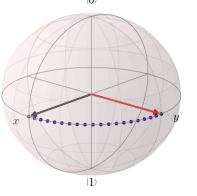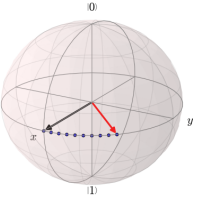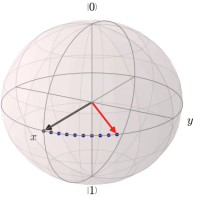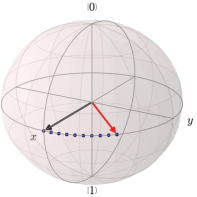
From Equ. 2.22, 2.23 and 2.24 it follows that X is its own inverse as well as its own Hermitian conjugate:

$$XX = XX^\dagger = \mathbb{1} \tag{2.25}$$
$$X = X^{-1} = X^\dagger \tag{2.26}$$

The action, circuit & matrix representation and Bloch sphere visualisation for the most important single-qubit quantum logic gates are summarised in Table 2.1 below.

Table 2.1: Table of major single-qubit quantum logic gates.

Gate	Name	Circuit representation	Matrix	Description	Rotation	Bloch sphere
$\mathbb{1}$	Identity	Id	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	Idle or waiting gate	-	-
X	Qubit flip	X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	Swaps amplitudes of $|0\rangle$ and $|1\rangle$	π rotation around $\hat{x}$	
Y	Qubit & phase flip	Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	Swaps amplitudes and introduces phase	π rotation around $\hat{y}$	
Z	Phase flip	Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Adds a negative sign to the $|1\rangle$ state	π rotation around $\hat{z}$	
H	Hadamard	H	$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$	Creates equal superpositions	$\frac{\pi}{2}$ rotation around $\hat{y}$ and π rotation around $\hat{x}$	
S	$\frac{\pi}{2}$ rotation gate	S	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\sqrt{Z}$	$\frac{\pi}{2}$ rotation around $\hat{z}$	
$S^\dagger$	$-\frac{\pi}{2}$ rotation gate	$S^\dagger$	$\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$	Adjoint of S	$-\frac{\pi}{2}$ rotation around $\hat{z}$	
T	$\frac{\pi}{4}$ rotation gate	T	$\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$	$\sqrt{S}$	$\frac{\pi}{4}$ rotation around $\hat{z}$	
$T^\dagger$	$-\frac{\pi}{4}$ rotation gate	$T^\dagger$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{-\frac{i\pi}{4}} \end{pmatrix}$	Adjoint of T	$-\frac{\pi}{4}$ rotation around $\hat{z}$	
ZM	Z-basis measurement		-	Measurement in standard basis	Collapses the state	-

2.2.2   Multi qubit gates

Controlled-NOT gate

The most important two-qubit quantum gate is the controlled NOT or CNOT gate given by the following 4x4 matrix:

$$CNOT = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.27}$$

The CNOT gate takes two qubits, a control and a target qubit, as input. If and only if the control qubit is in the $|1\rangle$ state, the NOT (X) gate is applied to the target qubit. In equations, the CNOT will always be followed by parentheses containing the control (c) qubit followed by the target (t) qubit: CNOT(c,t). The input-output relation (truth table) for the CNOT gate is given in Table 2.2 below.

Table 2.2: CNOT truth table with first qubit as control, second qubit as target.

Input	Output
00	00
01	01
10	11
11	10

To demonstrate the usefulness of the CNOT gate consider starting with two unentangled qubits both in the $|0\rangle$ state,

$$|\phi\rangle = |0\rangle \otimes |0\rangle = |00\rangle \tag{2.28}$$

Applying the H gate onto the first qubit yields the following (still unentangled) state:

$$(H \otimes \mathbb{1})\,|\phi\rangle = (H \otimes \mathbb{1})\,|00\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|10\rangle \tag{2.29}$$
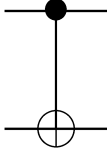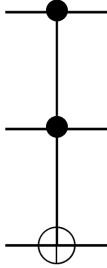
Now consider applying the CNOT to this state whereby the control qubit is coloured red and the target qubit green.

$$CNOT(0,1) \otimes (\frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|10\rangle) = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}(\mathbb{1} \otimes X)\,|10\rangle = \frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle \tag{2.30}$$

The last expression in Equ. 2.30 was used as an example (Equ. 2.19) for entanglement in Section 2.1.2 and it is one of the famous Bell states which are a set of four maximally entangled states (CITATION). Thus, this example demonstrates how the CNOT gate is crucial for the generation of entangled states since it applies the X gate to a qubit depending on the state of a second qubit.

The three most important multi-qubit quantum gates CNOT, Toffoli and nCNOT are characterised in Table 2.3.

Table 2.3: Table of major multi-qubit quantum logic gates where c_j stands for the j$^{\text{th}}$ control qubit and $\mathbb{1}_k$ for the kxk identity matrix.

Gate	Name	Circuit representation	Matrix	Description
CNOT	Controlled NOT		$\begin{pmatrix} \mathbb{1} & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	CNOT(c_1, target)
Toffoli	Controlled controlled NOT		$\begin{pmatrix} \mathbb{1}_6 & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$	CCNOT(c_1, c_2, target)
nCNOT	n-controlled NOT	-	$\begin{pmatrix} \mathbb{1}_{2^n-2} & 0 \\ 0 & X \end{pmatrix}$	nCNOT(c_1,..,c_n, target)

2.3   Classical machine learning

Machine learning (ML), a subfield of artificial intelligence, is aiming to enable computers to learn from data without a human explicitly programming its actions. It can be subdivided into the three major fields supervised & unsupervised machine learning and reinforcement learning. The three areas can be most readily understood by considering the following three colloquial sentences:

Supervised ML
- Based on *input* and *output* data
"I know how to classify this data but I need an algorithm to do the computations for me."

Unsupervised ML
- Based on *input* data only
"I have no clue how to classify this data, can the algorithm create a classifier for me?"

Reinforcement learning
- Based on *input* data only
"I have no clue how to classify this data, can the algorithm classify this data and I'll give it a reward if it's correct or I'll punish it if it's not."

This thesis, specifically, is focusing on quantum-enhancements in the field of supervised machine learning only. The following section will introduce a well-known algorithm from the subfield of supervised ML: the *k*-nearest neighbour algorithm.

2.3.1   Classical k-nearest neighbour algorithm

Understanding the quantum version of the distance weighted k-nearest neighbour (kNN) algorithm as proposed by Schuld, Sinayskiy, and Petruccione (2014) requires prior knowledge of the classical version of the algorithm that will be introduced in this subsection.

Imagine working for a search engine company and you are given the task of classifying unknown pictures of fruits as either apples or bananas. To train your classification algorithm, you are given five different pictures of apples and five different pictures of bananas. This will be called the *training data set* D_T. The pictures in D_T may be taken from various angles, in varying light settings and include different coloured apples and bananas. Two examples of such images are shown in Fig. 2.4.

Figure 2.4: Pictures of apple and banana from training data set D_T.[3]

Most of the time, using the full pixel representation of each picture for classification does not lead to optimal results. Therefore, the next step is to select a certain number of characteristic *features* extracted from the pictures in the training set that can be used to differentiate apples from bananas. Such a feature might be the RGB value of the most frequent pixel colour since bananas and apples have different colour spectra. Using a measure of the curvature of the main object in the picture is another possibility since an apple is almost spherical whereas a banana looks more like a bend line.

By selecting and extracting features, the dimensionality of the training data set is drastically reduced from a few thousand pixels to a handful of features. The m extracted features for the jth picture are stored in the m-dimensional *feature vector* $\vec{v}_j$. Mathematically, the training data set D_T consists of ten feature vectors $\vec{v}_0, \vec{v}_1, .., \vec{v}_{10}$ that are each assigned to either class A (apple) or B (banana). The training vectors are visualised as yellow and purple circles in Fig. 2.5.

Given a new picture of either a banana or an apple, you first extract the same m features from it and store it in the input vector $\vec{x}$. From many algorithms, you decide to use the kNN algorithm since it is a non-parametric classifier meaning it makes no prior assumption about the class of the new picture. Given a new unclassified input vector $\vec{x}$ (red star in Fig. 2.5), the algorithm considers the k nearest neighbours within the training set (using a predefined measure of distance) and classifies $\vec{x}$, based on a majority vote, as either A or B. Thereby, k is a positive integer, usually chosen to be small and its value determines the classification outcome. Namely, in the case $k = 3$ in Fig. 2.5, vector $\vec{x}$ will be classified as class B (purple) but in the case $k = 6$ it will be labelled class A (yellow).

---

[3]Reprinted from Pixabay, Retrieved December 24, 2016, from `https://pixabay.com/en/apple-red-fruit-frisch-vitamins-1632016/` and `https://pixabay.com/en/banana-fruit-yellow-1504956/`. Creative Commons Licence 2016 by Pixabay. Reprinted with permission.
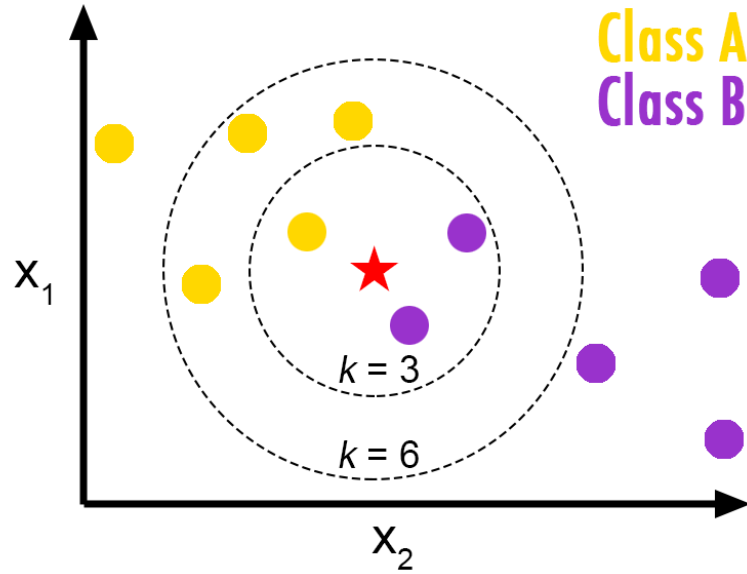
Figure 2.5: Visualization of a kNN classifier[4]

In the case of $k = 10$, $\vec{x}$ would simply be assigned to the class with the most members. In this case, the training vectors should be given distance-dependent weights (such as $\frac{1}{distance}$) to increase the influence of closer vectors over more distant ones.

2.3.2 Algorithmic time complexity

Big-O notation

In the fields of computer science and quantum information, the so-called Big-O notation, first described by Bachmann (1894), is often used to describe how the runtime of an algorithm depends on variables such as the desired accuracy, the number of input vectors or their size.

Definition: Big-O $\mathcal{O}$

For any monotonic functions $t(n)$ and $g(n)$ defined on a subset of the real numbers ($\mathbb{R}$), one says that $t(n) = \mathcal{O}(g(n))$ if and only if when there exists constants $c \geq 0$ and $n_0 \geq 0$ such that

$$| f(n) | \quad \leq \quad c * | g(n) | \quad \text{for all } n \geq n_0 \tag{2.31}$$

---

[4]Reprinted from GitHub, Burton de Wilde, Retrieved September 13, 2016, from `http://bdewilde.github.io/blog/blogger/2012/10/26/classification-of-hand-written-digits-3/`. Copyright 2012 by Burton de Wilde. Reprinted with permission.
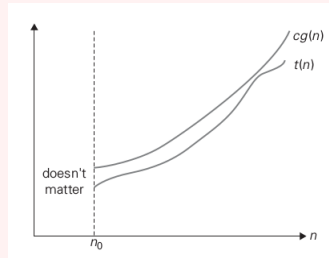
Figure 2.6: Visualization of $c * g(n)$ being an upper asymptotic bound of $t(n)$[5]

This implies that the function $f(n)$ does not grow at a faster rate than $g(n)$, or in other words that some constant multiple of the function $g(n)$ is an upper asymptotic bound for the function $f(n)$, for all $n \to \infty$.

[e]Reprinted from Anany Levitin and Soumen Mukherjee. Introduction to the Design & Analysis of Algorithms. Reading, MA: Addison-Wesley, 2003. Copyright 2012 by Levitin & Mukherjee.

Fig. 2.7 provides a good visual comparison between common algorithmic complexity classes regarding their relation to the input size n. It can be seen that the best possible algorithmic time complexity is constant time, $\mathcal{O}(1)$, that is being independent of the size of the input data e.g. determining if a binary number is even or odd. An algorithm is still considered excellent if it runs in logarithmic time $\mathcal{O}(log(n))$. Linear search algorithms, for example, are linear in time expressed as $\mathcal{O}(n)$. More complex sort algorithms like "bubble sort" have a higher time complexity and often run in quadratic time ($\mathcal{O}(n^2)$). Furthermore, algorithms with complexity $\mathcal{O}(n^c)$ are said to run in polynomial time for some $c > 2$. If an algorithm has time complexity $\mathcal{O}((log(n))^c)$ for some $c \geq 1$ it is called polylogarithmic. The highest complexity classes are exponential $\mathcal{O}(c^n)$ with $c \geq 1$ and factorial time $\mathcal{O}(n!)$. For example, solving the travelling salesman problem using brute-force search runs in factorial time.
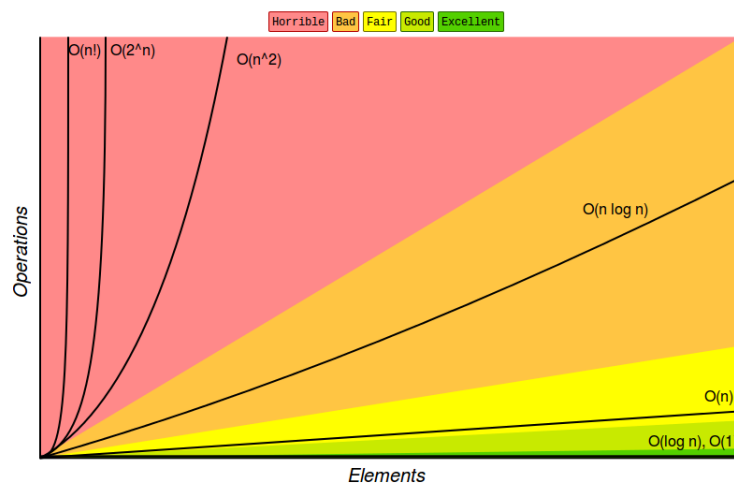
Figure 2.7: Overview of algorithmic complexity classes[6]

[6]Reprinted from Big-O Cheatsheet, Retrieved December 28, 2016, from `http://bigocheatsheet.com/`. Copyright 2016 by Big-O Cheatsheet. Reprinted with permission.

Chapter 3

Methods

The entirety of the research for this thesis is performed with pen and paper and a computer running the Linux operating system Ubuntu. The programming language Python, with its very intuitive syntax and large libraries for scientific computing and plotting, was used for the calculations and most of the plots for this thesis. More specifically, the open-source QuTiP library[7] for Python was used to create the Bloch sphere plots. All function plots were embedded directly into LaTeXby using the pgfplots package[8].

For the implementation of the quantum kNN algorithm there are two fundamentally different ways: Running it a) by simulating a QC or b) by actually executing it on a real QC. The required tools for both possibilities will be explained in the following subsections.

3.1 Liqui$|\rangle$

Classical computers can be used to simulate the behaviour of small quantum computers. Such simulations are associated with exponential computational costs thereby limiting the number of simulated qubits. Since current state-of-the-art quantum technology uses around ten qubits, a classical computer can still be used for simulation.

For the quantum computing simulations in this thesis the quantum simulation toolsuite Liqui$|\rangle$ developed by Wecker and Svore (2014) will be used. Liqui$|\rangle$ is based on the functional programming language F# and allows for simulation of up to 30 qubits (Microsoft Research, 2016). It comes with a large palette of predefined single and multi qubit quantum logic gates and allows for custom-defined quantum gates such as nCNOT and rotation gates controlled by n qubits which is crucial for some of the work done in this thesis. A short piece of example code from Liqui$|\rangle$ written in F# is shown in Fig. 3.1. For all quantum simulations in this thesis, a Lenovo ThinkPad T450 with an Intel i5 processor (2 cores) and 8GB random-access memory (RAM) is used.

---

[7]The open-source QuTiP library for Python may be downloaded from `http://qutip.org/`.
[8]The LaTeXpgfplots package may be downloaded from `https://www.ctan.org/pkg/pgfplots`.

```
275     [<LQD>] //defines that function can be called from the command line
276     let __LiquidCodeExample() = //define function name
277
278         show "This is a simple example of a Liqui|> function"
279
280         //initialize state vector (Dirac ket) with two qubits
281         let k  = Ket(2)
282         //create qubit list from ket k
283         let qs = k.Qubits
284
285         //apply Hadamard to first qubit in qubit list qs
286         H [qs.[0]]
287         //apply CNOT with control qs.[0] and target qs.[1]
288         CNOT [qs.[0];qs.[1]]
289         //measure the first qubit
290         M   [qs.[0]]
291         //measure the second qubit
292         M   [qs.[1]]
293
294         show "Measurement result for first qubit: %i" qs.[0].Bit.v
295         show "Measurement result for second qubit: %i" qs.[1].Bit.v
```

Figure 3.1: F# code snippet from Microsoft's quantum simulation toolsuite Liqui$|\rangle$

3.2   IBM Quantum Experience

Earlier this year IBM has enabled public cloud access to their experimental quantum processor containing five non error-corrected superconducting qubits located at the IBM Quantum Lab at the Thomas J Watson Research Center in Yorktown Heights, New York (IBM, 2016a). Instead of only simulating on classical hardware, this opens up the possibility of executing the QML algorithm on actual quantum hardware.

The so-called IBM Quantum Experience[9] provides the user with access to their *quantum composer* which is the main tool for algorithm design. The quantum composer shown in Fig. 3.2 consists of 5 horizontal lines, one for each qubit, and enables the user to choose from a universal gate set (bottom of Fig. 3.2) consisting of the following 10 quantum logic gates: $\mathbb{1}$, X, Y, Z, H, S, S†, T, T† and CNOT. Additionally, there are two different types of measurement: a) A measurement in the standard z-basis ($|0\rangle$ / $|1\rangle$) resulting in a probability distribution over all possible states and b) a Bloch measurement that visually projects the state onto the Bloch sphere. The user can compose an algorithm by applying up to 40 quantum logic and measurement gates to the five qubits by means of drag-and-drop.

---

[9]The IBM Quantum Experience can be accessed via `https://quantumexperience.ng.bluemix.net/qstage/`.
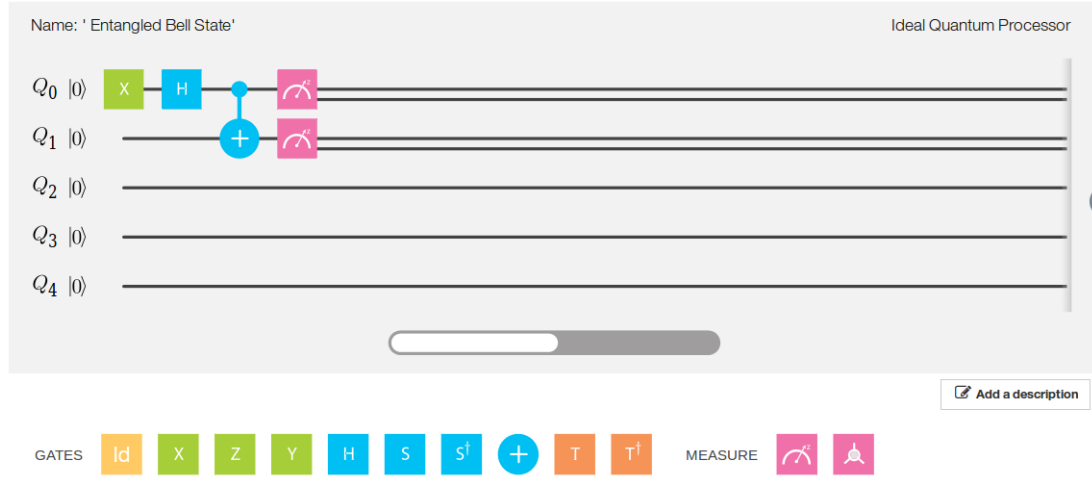
Figure 3.2: Screenshot showing IBM Quantum Composer[10]

By spending limited user coins the gate sequence of a composed algorithm is then send to IBM's QC in New York and depending on the waiting queue and the availability of the QC the results will be sent back via mail within a few minutes or days. IBM Quantum Experience also allows for free quantum simulations under ideal or real conditions which provides a great tool for experimentation without spending user coins.

The main limitation of the IBM Quantum Experience are the qubit decoherence times since they restrict the maximum number of possible operations before the qubits lose their quantum behaviour and their quantum information. Thus, the number of quantum gates is currently limited to only 40 which essentially means 39 logic gates and 1 measurement gate. According to the qubit calibration results shown in Fig. 3.3, the amplitude damping times of the five qubits range from 52.3 µs to 81.5 µs. Furthermore, the phase damping times range from 60.9 µs to 112.4 µs. Currently, the implementation of a single qubit quantum logic gate takes 130ns and applying a CNOT gate takes 500ns (Bishop, 2016).
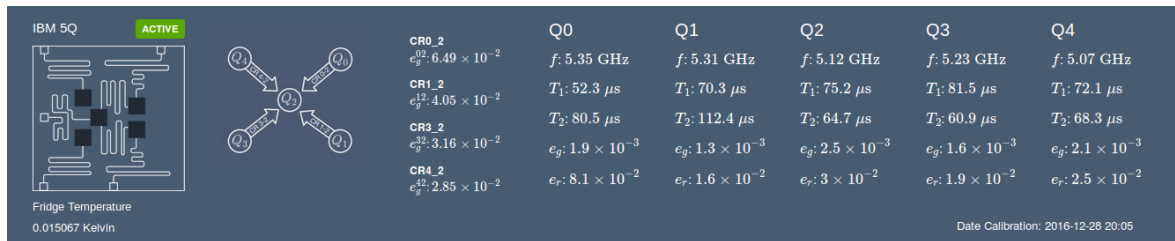
Figure 3.3: Screenshot of IBM QC calibration results (28. December 2016 - 20:05)[11]

---

[10]Screenshot was taken from `https://quantumexperience.ng.bluemix.net/qstage/#/editor`.
[11]Screenshot was taken from `https://quantumexperience.ng.bluemix.net/qstage/#/editor`.

Chapter 4

Literature Review: Quantum-enhanced Machine Learning

Classical machine learning takes classical data as input and learns from it using classical algorithms executed on classical computers - this will be referred to as C/C (classical data with classical algorithm). One enters the field of quantum machine learning when either quantum data or quantum algorithms are combined with ideas from classical machine learning. Thus, quantum machine learning can be subdivided into three different subfields: 1) C/Q - classical data with quantum algorithm, 2) Q/C - quantum data with classical algorithm and 3) Q/Q - quantum data with quantum algorithm.

Quantum data includes any data describing a quantum mechanical system such as e.g. the Hamiltonian of a system or the state vector of a certain quantum state. A *quantum algorithm* is any algorithm that can be executed on a quantum computer only.

The subfield Q/C processes quantum data with classical machine learning algorithms e.g. Las Heras, Alvarez-Rodriguez, Solano, and Sanz (2016) made use of classical genetic algorithms to improve the experimental and digital errors in quantum gates. Subfield Q/Q is the union of C/Q and Q/C and deals with the processing of quantum data using quantum algorithms e.g. learning the Hamiltonian of a quantum system using quantum machine learning algorithms.

The topic of this thesis is embedded within the subfield C/Q that aims to develop quantum algorithms for machine learning tasks involving classical data. This subfield is also called *quantum-enhanced machine learning* since clever algorithm design can harness quantum parallelism to speed-up classical machine learning algorithms. The following sections will introduce main concepts from the field of quantum-enhanced machine learning. More specifically, Section ?? will outline how classical data can be transferred into quantum states and Section ?? introduces a quantum version of the classical k-nearest neighbour algorithm.

4.1   Quantum state preparation

Quantum state preparation is the process of preparing a quantum state that accurately represents a vector containing classical (normalized) data. There are two fundamentally different ways of preparing a quantum state representing the classical example vector v:

$$v = \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \tag{4.1}$$

4.1.1   Encoding classical data into qubits

The most straightforward type of quantum state preparation does not make use of quantum superpositions but only uses the definite $|0\rangle$ or $|1\rangle$ states to store binary information in a multi qubit system as outlined in the example below.

Multiply vector v by ten such that the normalized entries can easily be represented in binary,

$$\begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} * 10 = \begin{pmatrix} 6 \\ 4 \end{pmatrix} \tag{4.2}$$

Convert each entry to binary,

$$\begin{pmatrix} 6 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 0110 \\ 0100 \end{pmatrix} \tag{4.3}$$

Rewrite the 2-D vector as a 1-D bit string,

$$\begin{pmatrix} 0110 \\ 0100 \end{pmatrix} \rightarrow n = 01100100 \tag{4.4}$$

For g bits initialize g qubits in the $|0\rangle$ state & apply the X gate to the respective qubits:

$$n = 01100100 \rightarrow |n\rangle = (\mathbb{1} \otimes X \otimes X \otimes \mathbb{1} \otimes \mathbb{1} \otimes X \otimes \mathbb{1} \otimes \mathbb{1}) |00000000\rangle = |01100100\rangle \tag{4.5}$$

When encoding classical data into qubit states, a k-dimensional probability vector requires $4k$ classical bits which are encoded one-to-one into $4k$ qubits. Thus, the number of qubits increases linearly with the size of the classical data vector. Due to this one-to-one correspondence between classical bits and qubits there is no data compression improvement compared to classical data storage.

Qubit-based quantum state preparation becomes slightly more complicated when aiming to achieve a quantum memory state $|M\rangle$ in an equal superposition of l binary patterns l^j of the form:

$$|M\rangle = \frac{1}{\sqrt{l}} \sum_{j=1}^{l} |l^j\rangle \text{ where } |l^j\rangle = |l_1^j, l_2^j, ... l_n^j\rangle \text{ and } l_k^j \in \{0, 1\} \tag{4.6}$$

Preparing this quantum state is a requirement for the later used qubit-encoded kNN quantum algorithm by Schuld et al. (2014). Trugenberger (2001) describe a quantum routine that can efficiently prepare such a state as will be explained in detail below.

First, Trugenberger (2001) defines the new unitary quantum gate S^j,

$$S^j = \begin{pmatrix} \sqrt{\frac{j-1}{j}} & \frac{1}{\sqrt{j}} \\ -\frac{1}{\sqrt{j}} & \sqrt{\frac{j-1}{j}} \end{pmatrix} \tag{4.7}$$

and introduces its controlled version CS^j:

$$CS^j = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & S^j \end{pmatrix} \tag{4.8}$$

The initial quantum state is given in Equ. 4.9 and consists of three registers; the first being the pattern register containing the first pattern l^1, the second register u is a utility register initialized in state $|01\rangle$ and the third register m represents the memory register initialized with n zeros in which all patterns l^j will be loaded one after the other.

$$|\Psi_0^1\rangle = |l^1; u; m\rangle = |l_1^1, l_1^1, ..., l_n^1; 01; 0_1, ..., 0_n\rangle \tag{4.9}$$

The routine will use the second utility qubit u_2 to separate the intial state into two terms whereby $u_2 = |0\rangle$ flags the already stored patterns and $u_2 = |1\rangle$ indicates the processing term. In order to store a pattern l^j in the memory register one has to perform the following operations:

Step 1: Using u_2 as one of the control qubits for the CCNOT gate, copy the pattern l^j into the memory register of the processing term ($u_2 = |1\rangle$):

$$|\Psi_1^j\rangle = \prod_{r=1}^{n} CCNOT(l_r^j, u_2, m_r) |\Psi_0^j\rangle \tag{4.10}$$

Step 2: If the qubits in the pattern and memory register are identical (true only for the processing term) then overwrite all qubits in the memory register with ones:

$$|\Psi_2^j\rangle = \prod_{r=1}^{n} X(m_r) CNOT(l_r^j, m_r) |\Psi_1^j\rangle \tag{4.11}$$

Step 3: Apply a nCNOT gate controlled by all n qubits in the m register and flip u_2 if and only if all n qubits are ones (true only for the processing term):

$$|\Psi_3^j\rangle = nCNOT(m_1, m_2, ..., m_n, u_2) |\Psi_2^j\rangle \tag{4.12}$$

Step 4: Using the previously defined CS^j operation, with control u_1 and target u_2, the new pattern is transferred from the processing term into the term containing the already stored patterns ($u_2 = |0\rangle$):

$$|\Psi_4^j\rangle = CS^{l+1-j}(u_1, u_2) |\Psi_3^j\rangle \tag{4.13}$$

Step 5 & 6: In Step 2 & 3 all qubits in the memory register were overwritten with ones and these steps can be undone by applying their inverse operations:

$$|\Psi_5^j\rangle = nCNOT(m_1, m_2, ..., m_n, u_2)|\Psi_4^j\rangle \tag{4.14}$$

$$|\Psi_6^j\rangle = \prod_{r=n}^{1} CNOT(l_r^j, m_r)X(m_r)|\Psi_5^j\rangle \tag{4.15}$$

The resulting state is now given by the following equation:

$$|\Psi_6^j\rangle = \frac{1}{\sqrt{l}}\sum_{w=1}^{j}|l^j; 00; l^w\rangle + \sqrt{\frac{l-j}{l}}|l^j; 01; l^j\rangle \tag{4.16}$$

Step 7: Finally, by applying the inverse operation of Step 1 the memory register of the processing term is restored to zeros only:

$$|\Psi_7^j\rangle = \prod_{r=n}^{1} CCNOT(l_r^j, u_2, m_r)|\Psi_6^j\rangle \tag{4.17}$$

At the end of Step 7 the next pattern can be loaded into the first register and by applying Steps 1-7 again the pattern gets added to the memory register. After repeating this procedure l times the memory register m will be in the desired state $|M\rangle$ defined by Equ. 4.6.

4.1.2   Encoding classical data into amplitudes

A more sophisticated way of representing the classical vector v (Equ. 4.1) as a quantum state makes use of the large number of available amplitudes in a multi qubit system. The general idea is demonstrated in Equ. 4.18 below.

$$\begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \quad \rightarrow \quad |n\rangle = \sqrt{0.6}|0\rangle + \sqrt{0.4}|1\rangle \tag{4.18}$$

Using amplitude-based quantum state preparation, a k-dimensional probability vector is encoded into only $log_2(k)$ qubits since the number of amplitudes grows exponentially with the number of qubits. This type of quantum data storage makes exponential compression of classical data possible. Since a quantum gate acts on all amplitudes in the superposition at once there is the possibility of exponential speed-ups in quantum algorithms compared to their classical counterparts. Compared to the one-to-one correspondence in qubit-encoded state preparation, amplitude-encoding requires a much smaller number of qubits that grows logarithmically with the size of the classical data vector. However, initializing an arbitrary amplitude distribution is still an active field of research and requires the implementation of non-trivial quantum algorithms.

For the case when the classical data vectors represent discrete probability distributions which are efficiently integrable on a classical computer, Grover and Rudolph (2002) developed a quantum routine to initialize the corresponding amplitude distribution. Additionally, Soklakov and Schack (2006) proposed a quantum algorithm for the more general case that includes initializing amplitude distributions for classical data vectors representing non-efficiently integrable probability distributions.

4.2 Quantum k-nearest neighbour algorithm

The quantum distance-weighted kNN algorithm outlined in this section was proposed by Schuld et al. (2014) and is based on classical data being encoded into qubits rather than amplitudes. The first step is to prepare an equal superposition $|T\rangle$ over N training vectors $\vec{v}$ of length n with binary entries $v_1, v_2, ... v_n$ each assigned to a class c as follows,

$$|T\rangle = \frac{1}{\sqrt{N}} \sum_{p=1}^{N} |v_1^p, v_2^p, ... v_n^p; c^p\rangle \tag{4.19}$$

The unknown vector $\vec{x}$ of length n and binary entries $x_1, x_2, ... x_n$ needs to be classified and is added to the training superposition resulting in the initial state $|\psi_0\rangle$:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{p=1}^{N} |x_1, x_2, ... x_n; v_1^p, v_2^p, ... v_n^p; c^p\rangle \tag{4.20}$$

An ancilla qubit initially in state $|0\rangle$ is added to the state such that the superposition is now described by,

$$|\psi_1\rangle = \frac{1}{\sqrt{N}} \sum_{p=1}^{N} |x_1, x_2, ... x_n; v_1^p, v_2^p, ... v_n^p; c^p\rangle \otimes |0\rangle \tag{4.21}$$

The state now consists of four registers: 1) input(x) register, 2) training(v) register, 3) class(c) register and 4) ancilla($|0\rangle$) register. Next, the ancilla register is put into an equal superposition by applying an H gate to it,

$$|\psi_2\rangle = \frac{1}{\sqrt{N}} \sum_{p=1}^{N} |x_1, x_2, ... x_n; v_1^p, v_2^p, ... v_n^p; c^p\rangle \otimes H|0\rangle$$

$$= \frac{1}{\sqrt{N}} \sum_{p=1}^{N} |x_1, x_2, ... x_n; v_1^p, v_2^p, ... v_n^p; c^p\rangle \otimes \frac{(|0\rangle + |1\rangle)}{\sqrt{2}} \tag{4.22}$$

The main step in any kNN algorithm is calculating some measure of distance between each training vector $\vec{v}$ and the input vector $\vec{x}$ which in this quantum algorithm is taken to be the Hamming distance defined in the red box below.

Definition: Hamming distance

First defined by Hamming (1950), Hamming distance (HD) is the number of differing characters when comparing two equally long binary patterns p_0 and p_1.

Example:
$p_0 =$ 0   0   1
$p_1 =$ 1   0   1
$- - - - - - - -$
HD $= 1 + 0 + 0 = 1$ According to Trugenberger (2001), HD is the squared Euclidean distance between the two binary patterns p_0 and p_1.

Given quantum state $|\psi_2\rangle$ the HD between the input and each training register can be calculated by applying $\text{CNOT}(x_s, v_s^p)$ gates to all qubits in the first and second register using the input vector qubits x_s as controls and the training vector qubits v_s^p as targets. The sum of the qubits in the second register now represents the total HD between each training register and the input. Applying an X gate to each qubit in the second register reverses the HD such that small HDs become large

and vice versa. This is crucial since training vectors close to the input should get larger weights than more distant vectors. This procedure will change the qubits in the second register according to the rules specified in Equ. 4.23.

$$d_s^p = \begin{cases} 1, & \text{if } |v_s^p\rangle = |x_s\rangle \\ 0, & \text{otherwise} \end{cases} \tag{4.23}$$

The quantum state is now given by,

$$
\begin{aligned}
|\psi_3\rangle &= \prod_{s=1}^{n} X(v_s^p) CNOT(x_s, v_s^p) |\psi_2\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{p=1}^{N} |x_1, x_2, ...x_n; d_1^p, d_2^p, ...d_n^p; c^p\rangle \otimes \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}
\end{aligned}
\tag{4.24}
$$

By applying the unitary operator U defined by,

$$U = e^{-i\frac{\pi}{2n}K} \tag{4.25}$$

where

$$K = \mathbb{1} \otimes \sum_s (\frac{\sigma_z + 1}{2})_{d_s} \otimes \mathbb{1} \otimes (\sigma_z)_c \tag{4.26}$$

the sum over the second register is computed. As a result, the total reverse HD, denoted $d_H(\vec{x}, \vec{v}^p)$, between the pth training vector $\vec{v}^p$ and the input vector $\vec{x}$ is written into the complex phase of the pth term in the superposition. The ancilla register is now separating the superposition into two terms due to a sign difference in the amplitudes (negative sign when ancilla is $|1\rangle$). The result is given by,

$$
\begin{aligned}
|\psi_4\rangle &= U |\psi_3\rangle \\
&= \frac{1}{\sqrt{2N}} \sum_p^N e^{i\frac{\pi}{2n}d_H(\vec{x}, \vec{v}^p)} |x_1, x_2, ...x_n; d_1^p, d_2^p, ...d_n^p; c^p; 0\rangle \\
&\quad + e^{-i\frac{\pi}{2n}d_H(\vec{x}, \vec{v}^p)} |x_1, x_2, ...x_n; d_1^p, d_2^p, ...d_n^p; c^p; 1\rangle
\end{aligned}
\tag{4.27}
$$

Applying an H gate to the ancilla register transfers the $d_H(\vec{x}, \vec{v}^p)$ from the phases into the amplitudes such that the new quantum state is described by,

$$
\begin{aligned}
|\psi_5\rangle &= (\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes H) |\psi_4\rangle \\
&= \frac{1}{\sqrt{N}} \sum_p^N cos\big[\frac{\pi}{2n}d_H(\vec{x}, \vec{v}^p)\big] |x_1, x_2, ...x_n; d_1^p, d_2^p, ...d_n^p; c^p; 0\rangle \\
&\quad + sin\big[\frac{\pi}{2n}d_H(\vec{x}, \vec{v}^p)\big] |x_1, x_2, ...x_n; d_1^p, d_2^p, ...d_n^p; c^p; 1\rangle
\end{aligned}
\tag{4.28}
$$

At this point the ancilla qubit is measured along the standard basis and all previous steps have to be repeated until the ancilla is measured in the $|0\rangle$ state. Since it is conditioned on a particular outcome, this type of measurement is called *conditional measurement* (CM). The probability of a successful CM is given by the square of the absolute value of the amplitude and is dependent on the average reverse HD between all training vectors and the input vector:

$$Prob(|a\rangle = |0\rangle) = \sum_p^N cos^2\big[\frac{\pi}{2n}d_H(\vec{x}, \vec{v}^p)\big] \tag{4.29}$$

Finally, to classify the input vector $\vec{x}$ the class register is measured along the standard basis. The probability of measuring a specific class c is then given by the following expression:

$$Prob(c) = \frac{1}{NProb(|a\rangle = |0\rangle)} \sum_{l \in c} cos^2 \left[\frac{\pi}{2n} d_H(\vec{x}, \vec{v}^l) \right] \qquad (4.30)$$

From Equ. 4.30 it is evident that the probability of measuring a certain class c is dependent on the average total reverse HD between all training vectors belonging to class c and the input vector $\vec{x}$. Since the total reverse HD represent distance-dependent weights it gets clear why this is the quantum equivalent to a classical distance-weighted kNN algorithm.

In order to obtain a full picture of the probability distribution over the different classes a sufficient number of copies of $|\psi_5\rangle$ needs to be prepared and after successful CM on the ancilla qubit the class qubit needs to be measured.

Complexity analysis

According to Schuld et al. (2014), the preparation of the superposition has a complexity of $\mathcal{O}(Pn)$ where P is the number of training vectors and n is the length of the feature vectors. The algorithm has to be repeated T times in order to get a statistically precise picture of the results. Hence, the total quantum kNN algorithm has an algorithmic complexity of $\mathcal{O}(TPn)$.

Chapter 5

Results and Discussion

This chapter is subdivided into two sections; Section 5.1 focuses on the simulation of the qubit-based kNN algorithm proposed by Schuld et al. (2014) and Section 5.2 introduces a newly developed amplitude-based kNN algorithm with an implementation on IBM's QC in mind. Section 5.2 is then further subdivided into an attempt to compile and implement the algorithm using the IBM Quantum Experience (Section 5.2.1) and its simulation using Liqui|⟩ (Section 5.2.2). All the F# code written for the quantum simulations described in this chapter can be found on GitHub[12].

5.1 Simulating the qubit-based kNN algorithm

Similar to classical machine learning, the first step towards simulating any quantum machine learning algorithm is the careful selection of a suitable classification problem. The computer used for the Liqui|⟩ quantum simulations in this thesis only provides 8GB of RAM, thereby limiting the maximum number of simulated qubits to 24. Unfortunately, real-world machine learning problems usually involve large datasets that would require much more qubits such that a small artificial dataset needs to be constructed. For this reason, the classification of 9-bit, little-endian RGB colour codes into the classes *red* and *blue* will be considered. A 9-bit RGB colour code uses three bits to encode the content of each RGB colour; red, green and blue. Three binary bits b_0, b_1, b_2 can encode any of the numbers 0-7 according to the formula,

$$b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^2 \text{ where } b_0, b_1, b_2 \in \{0, 1\} \tag{5.1}$$

For example, the 9-bit RGB code 111 100 100 can be written in roman numerals as 7,1,1 (full red, little green, little blue) and represents this red tone .

The quantum kNN algorithm is evaluated on two different levels of difficulty - easy and hard. First, the algorithm will be tested using training set I consisting of 3 randomly chosen red and blue tones listed in Table 5.1. Since the class qubit $|c\rangle$ can only take binary values, class red is defined as $|c\rangle = |0\rangle$ and class blue is defined as $|c\rangle = |1\rangle$. Note that this assessment stage is considered easy because all training colours are either pure red colours (without green or blue content) or pure blue colours (without green or red content).

The quantum classifier will then be tested on four new colour tones (two red, two blue) listed in Table 5.1. However, in contrast to the training set, the input colours also include colours with additional green content testing how the classifier reacts to cases that it has not been trained for.

---

[12]The code is published under an open-source licence and can be downloaded from `https://github.com/markf94/Liquid_QML`.

Colour	Binary 9-bit RGB string	Class
	111 000 000	red ($|0\rangle$)
	101 000 000	red ($|0\rangle$)
	110 000 000	red ($|0\rangle$)
	000 000 111	blue ($|1\rangle$)
	000 000 101	blue ($|1\rangle$)
	000 000 100	blue ($|1\rangle$)

Table 5.1: Training data set I

Colour	Binary 9-bit RGB string	Expected class
	100 000 000	red ($|0\rangle$)
	110 100 000	red ($|0\rangle$)
	000 000 110	blue ($|1\rangle$)
	000 100 111	blue ($|1\rangle$)

Table 5.2: Input data set I

In the more difficult evaluation stage, two more blue and red colours are added to training data set I resulting in training data set II shown in Table 5.1. Note that the four new training colours have mixed colour contents meaning red training colours might have a slight blue or green content and vice versa.

In order to compare how training data set I and II influence the classification outcome the four colours from input data set I are also present in input data set II listed in Table 5.1. Additionally, four new colour tones were added that consist of different mixtures of red, green and blue. These new input colours constitute interesting edge cases and their correct classification is considered hard. Note that the fourth colour from the top in Table 5.1 has equal red and blue content and will be used to test the classifier in an indecisive case.

Colour	Binary 9-bit RGB string	Class
	111 000 000	red ($|0\rangle$)
	101 000 000	red ($|0\rangle$)
	110 000 000	red ($|0\rangle$)
	111 100 100	red ($|0\rangle$)
	111 000 100	red ($|0\rangle$)
	000 000 111	blue ($|1\rangle$)
	000 000 101	blue ($|1\rangle$)
	000 000 100	blue ($|1\rangle$)
	100 000 111	blue ($|1\rangle$)
	100 110 111	blue ($|1\rangle$)

Table 5.3: Training data set II

Colour	Binary 9-bit RGB string	Expected class
	100 000 000	red ($|0\rangle$)
	110 100 000	red ($|0\rangle$)
	101 100 100	red ($|0\rangle$)
	110 100 110	none
	000 000 110	blue ($|1\rangle$)
	000 100 111	blue ($|1\rangle$)
	100 100 111	blue ($|1\rangle$)
	110 100 101	blue ($|1\rangle$)

Table 5.4: Input data set II

The first step towards the classification of RGB colour codes using the qubit-based kNN algorithm proposed by Schuld et al. (2014) as described in detail in Section ?? is preparing the initial superposition over all six 9-bit RGB training patterns t^j:

$$|T\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{N} |t_1^j, t_2^j, ...t_9^j; c^j\rangle \tag{5.2}$$

This can be done using the quantum state preparation algorithm by Trugenberger (2001) outlined in Section 4.1.1. In this case, the seven steps (see blue box in Section 4.1.1) of the state preparation

algorithm have to be repeated six times in order to load the six RGB training patterns t^j into the memory register m defined in Equ. 4.9. Afterwards, the state is given by,

$$|\phi_0\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |t_1^6, t_2^6, ..., t_9^6; u_1 = 0, u_2 = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{5.3}$$

Equ. 5.3 shows that the first register still contains the last stored RGB colour code t^6, the second register consists of the utility qubits $|u_1\rangle$ and $|u_2\rangle$ that are both in the $|0\rangle$ state and the last register is in an equal superposition over all six RGB training colours. Thus, besides the missing class qubit $|c\rangle$ the last register is in the desired superposition defined in Equ. 5.2.

In the next step of the quantum kNN the yet unclassified 9-bit RGB input pattern x and an ancilla qubit $|a\rangle$ initialized in state $|0\rangle$ are added to the training superposition $|T\rangle$ to result in the full initial state $|\psi_0\rangle$:

$$|\psi_0\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |x_1, x_2, ...x_9; t_1^j, t_2^j, ...t_9^j; c^j; 0\rangle \tag{5.4}$$

The trick now is to realize that Equ. 5.3 and 5.4 contain the same number of qubits. Firstly, each of them has nine qubits in the first register. Secondly, Equ. 5.3 has two utility qubits that are balanced by the ancilla and the class qubit in Equ. 5.4. Lastly, there are again nine qubits in the third register in Equ. 5.3 and the second register in Equ. 5.4. Thus, $|\psi_0\rangle$ and $|\phi\rangle$ both contain $9+9+2 = 20$ qubits. Since $|\phi\rangle$ is the current state of the qubits in the simulation, one simply needs to redefine the utility qubits such that the first one becomes the class qubit $|u_1\rangle = |c^j\rangle$ and the second utility qubit becomes the ancilla $|u_2\rangle = |a\rangle$. Note that class and ancilla qubit are currently still in the $|0\rangle$ state as indicated in the current quantum state $|\phi_1\rangle$:

$$|\phi_1\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |t_1^6, t_2^6, ..., t_9^6; c^j = 0; a = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{5.5}$$

Equ. 5.5 shows a quantum state with four registers as desired but the first register still contains the last training colour code t^6. However, it can simply be overwritten with the desired input pattern by comparing the two patterns and flipping qubits at positions where the patterns do not match up. For example, if the last training pattern was 000 000 100 and the input pattern is 000 000 110 one simply needs to flip the 8$^{\text{th}}$ qubit through the application of an X gate. The state is now given by,

$$|\phi_2\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |x_1, x_2, ..., x_9; c^j = 0; a = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{5.6}$$

Currently, every training pattern is considered class $|0\rangle$ (red) which is, of course, incorrect. To flip the class qubit for the three training patterns encoding blue colours, one can make use of X and nCNOT gates. Consider the fourth (first blue) training pattern $t^4 = 000\ 000\ 111$ from Table **??**. One might think that simply applying a 3CNOT(t_7, t_8, t_9,c) gate controlled by the three qubits that are in the $|1\rangle$ state will suffice to flip the class label for this training pattern. However, depending on the classification problem at hand there might be another training pattern e.g. $t' = 111\ 110\ 111$

belonging to class $|0\rangle$ for which the application of the 3CNOT(t_7, t_8, t_9,c) gate would incorrectly flip the class qubit since the last three qubits of t' are also in the $|1\rangle$ state.

To avoid this problem, apply an X gate to all qubits in the training pattern that are currently in the $|0\rangle$ state. Continuing the example with the training pattern $t^4 = 000\ 000\ 111$, X gates need to be applied to the first six qubits. The result is then $t^{4*} = 111\ 111\ 111$. After this step, any other training pattern will contain at least one zero e.g. flipping the first six qubits of $t' = 111\ 110\ 111$ results in $t'^* = 000\ 001\ 111$. Hence, the training pattern t^4 is now the only pattern in the fourth register consisting only of ones. Now, this property can be exploited by applying a 9CNOT($t_1, t_2, ..., t_9$,c) gate that will flip the class label to $|1\rangle$ for training pattern t^4 only. Acting X gates on the first six qubits again will restore all training patterns to their original states. Repeating this procedure for all training patterns belonging to class $|1\rangle$ (blue) entangles the class qubit $|c\rangle$ with the training patterns and the overall state is now described by,

$$|\phi_3\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |x_1, x_2, ..., x_9; c^j; a=0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{5.7}$$

Note that the class qubit is now not in the $|0\rangle$ state only. Inspection of Equ. 5.7 reveals that $|\phi_3\rangle$ is identical to the desired initial state $|\psi_0\rangle$ defined in Equ. 5.4 the only difference being the position of the class and ancilla register. One can now proceed with the quantum kNN routine by simply putting the ancilla register into superposition with an H gate.

$$|\phi_4\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \left[|x_1, x_2, ..., x_9; c^j; 0; t_1^j, t_2^j, ..., t_9^j\rangle + |x_1, x_2, ..., x_9; c^j; 1; t_1^j, t_2^j, ..., t_9^j\rangle \right] \tag{5.8}$$

The next step is the calculation of the HD between the input pattern and each training pattern which is done by the straightforward application of nine CNOT(x_s, t_s^j) gates. By applying an X gate to every qubit in the fourth register the HD gets reversed as discussed in Section **??**. The state is now,

$$|\phi_5\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \prod_{s=1}^{9} X(t_s^j) CNOT(x_s, t_s^j) \Big[|x_1, x_2, ..., x_9; c^j; 0; t_1^j, t_2^j, ..., t_9^j\rangle$$

$$+ |x_1, x_2, ..., x_9; c^j; 1; t_1^j, t_2^j, ..., t_9^j\rangle \Big] \tag{5.9}$$

$$= \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \left[|x_1, x_2, ..., x_9; c^j; 0; d_1^j, d_2^j, ...d_9^j\rangle + |x_1, x_2, ..., x_9; c^j; 1; d_1^j, d_2^j, ...d_9^j\rangle \right] \tag{5.10}$$

In order to perform the sum over the fourth register and store the result in the complex phase of the corresponding term in the superposition one needs to implement the unitary operator U previously defined in Equ. 4.25 and 4.26 with $n = 9$ in the case of 9-bit RGB classification. According to Trugenberger (2001) the operator U can be decomposed as follows:

$$U |\phi_5\rangle = e^{-i\frac{\pi}{2n}K} |\phi_5\rangle = e^{-i\frac{\pi}{18}K} |\phi_5\rangle = \prod_{f=1}^{9} CL^{-2}(a, t_f) \prod_{k=1}^{9} L(t_k) |\phi_5\rangle \tag{5.11}$$

where $L = \begin{pmatrix} e^{-i\frac{\pi}{18}} & 0 \\ 0 & 1 \end{pmatrix}$ and $CL^{-2} = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & L^{-2} \end{pmatrix}$ and $L^{-2} = \begin{pmatrix} e^{i\frac{\pi}{9}} & 0 \\ 0 & 1 \end{pmatrix}$

The unitary gates U, CU^{-2} and U^{-2} can easily be defined in Liqui|⟩'s programming environment and acting them on quantum state $|\phi_5\rangle$ leads to the result:

$$|\phi_6\rangle = U|\phi_5\rangle = \frac{1}{\sqrt{12}}\sum_{j=1}^{6}\left[e^{i\frac{\pi}{18}d_H(\vec{x},\vec{v}^p)}|x_1,x_2,...,x_9;c^j;0;d_1^j,d_2^j,...d_9^j\rangle\right.$$
$$\left. + e^{-i\frac{\pi}{18}d_H(\vec{x},\vec{v}^p)}|x_1,x_2,...,x_9;c^j;1;d_1^j,d_2^j,...d_9^j\rangle\right] \qquad (5.12)$$

In the last step, one simply has to act an H gate on the ancilla qubit in the third register which will transfer the total reverse HD from the phases into the amplitudes shown in Equ. **??** below.

$$|\phi_7\rangle = (\mathbb{1}\otimes\mathbb{1}\otimes H\otimes\mathbb{1})|\phi_6\rangle = \frac{1}{\sqrt{12}}\sum_{j=1}^{6}\left[cos\left[\frac{\pi}{18}d_H(\vec{x},\vec{t}^j)\right]|x_1,x_2,...,x_9;c^j;0;d_1^j,d_2^j,...d_9^j\rangle\right.$$
$$\left. + sin\left[\frac{\pi}{18}d_H(\vec{x},\vec{t}^j)\right]|x_1,x_2,...,x_9;c^j;1;d_1^j,d_2^j,...d_9^j\rangle\right] \qquad (5.13)$$

At this point, the ancilla qubit in the third register is conditionally measured. This can be achieved with a simple if statement in F# as shown in the pseudocode below:

```
if ancilla = 0 then
    measure class qubit
else
    start a new run
```

If and only if the ancilla is found to be in the $|0\rangle$ state, the class qubit is measured. The procedure is repeated for y runs to gather sufficiently accurate statistics. Finally, the input vector is assigned to the most frequently measured class.

Quantum simulation results

All the steps outlined in the previous section were programmed in F# within the Liqui|⟩ framework. When executing the script[13] the user needs to first specify the number of runs r needed to gather sufficiently accurate statistics. Next, the user is asked to input the total number of training patterns. One-by-one the user then manually inputs all binary training patterns and their respective classes. Lastly, the user is asked to specify the input pattern requiring classification. The algorithm will then be simulated and repeated r times.

The algorithm was first trained using the 9-bit RGB colours from training data set I and then asked to classify each input pattern from input data set I. Each simulation was repeated 100 times to gather sufficient statistics. The results are listed in Table 5.5. For every input pattern the probabilities of successful CM ($Prob(CM)$) and measuring the class qubit in either the $|0\rangle$ ($Prob$ ($|c\rangle = |0\rangle$)) or the $|1\rangle$ ($Prob$ ($|c\rangle = |1\rangle$)) state are shown.

For every input pattern and probability, the theoretical prediction (marked with asterisks) on top is contrasted with the simulation result below. The small differences between these numbers show that in most cases 100 runs suffice to retrieve the theoretically predicted probabilities. Yet, the first red input pattern 100 000 000 is the only exception since it was incorrectly classified as blue despite the theory predicting a slightly higher probability to measure class red ($|c\rangle = |0\rangle$). In this

---

[13]The script "__TrugenbergerSchuld(r)", where r is the number of runs, can be found in the */linux/bin/Release/* folder in the GitHub repository https://github.com/markf94/Liquid_QML.

Colour	Binary 9-bit RGB string	$Prob(CM)$	$Prob$ $(|c\rangle = |0\rangle)$	$Prob$ $(|c\rangle = |1\rangle)$	Expected class	Algorithm output
	100 000 000	0.8404* —— 0.7500	0.5598* —— 0.4933	0.4402* —— 0.5067	red ($|0\rangle$)	blue ($|1\rangle$)
	110 100 000	0.6421* —— 0.6200	0.6756* —— 0.6129	0.3244* —— 0.3871	red ($|0\rangle$)	red ($|0\rangle$)
	000 000 110	0.7349* —— 0.7000	0.3599* —— 0.3571	0.6401* —— 0.6429	blue ($|1\rangle$)	blue ($|1\rangle$)
	000 100 111	0.5366* —— 0.5300	0.1916* —— 0.0943	0.8084* —— 0.9057	blue ($|1\rangle$)	blue ($|1\rangle$)

Table 5.5: Classification results for input data set I after 100 runs. Trained with training data set I. Theoretical predictions (marked with asterisks) on top, simulation results at the bottom.

case, however, both class probabilities are almost equal such that even after a large number of runs the classification would not be better than a coin flip.

The other three input pattern from input data set I were all correctly classified. In all of these cases, the probabilities were clearly favouring one class over the other. It is important to note that the algorithm correctly classified the two input patterns with additional green colour content, even though it was only trained with pure red and pure blue colours.

Next, the algorithm was trained using training data set II and asked to classify all eight input pattern from input data set II, again repeating each simulation 100 times. The results for this second evaluation stage are listed in Table 5.6. In most cases, the probabilities retrieved from the simulations again closely resemble the theoretically predicted probabilities after 100 runs.

This time, the first red input pattern 100 000 000, previously incorrectly classified using training data set I, is correctly assigned to class red. This is a good example of how the size and quality of the training data set can influence the classification outcome. This goes as far that according to Domingos (2012) a simple classifier, e.g. kNN, trained on a large training set with correct labels often performs better than a more complicated classifier trained using the same training set.

The second red colour, with slight green content, was again correctly classified but this time the probability distribution over the class qubit states is different when compared with the corresponding entries in Table 5.5. Using training data set II instead of training set I the probability of measuring class $|0\rangle$ increased by 0.12480. This again underlines the importance of size and quality of the training data set.

Consisting of majority red with slight green and blue content, the third red colour was still classified correctly. Although, the probabilities of measuring class $|0\rangle$ and $|1\rangle$ are close to being equal the simulation yielded a distribution closely resembling the theoretical prediction after only 100 runs.

The fourth colour from the top in Table 5.6 with a slight green and equal red and blue content was deliberately chosen as an edge case. The classifier assigned it to class blue but the class probabilities obtained from the simulation are equal for the $|0\rangle$ and $|1\rangle$ state. Interestingly, the theoretical probabilities slightly favour class red. This can be traced back to the fact that the quantum kNN is based on distance-weighting. This can be seen when analysing training data set II; all but one red training colour have a 1 at the second bit position whereas all but two blue training colours have a 1 at the second to last bit position. Since the fourth input colour has a 1 at the second as well as at the second to last bit position the HDs to the red training colours are slightly smaller than to the blue training colours. Since the HDs are reversed the red class is

Colour	Binary 9-bit RGB string	$Prob(CM)$	$Prob$ $(\lvert c\rangle = \lvert 0\rangle)$	$Prob$ $(\lvert c\rangle = \lvert 1\rangle)$	Expected class	Algorithm output
	100 000 000	0.7543* / 0.8500	0.5515* / 0.5529	0.4485* / 0.4471	red ($\lvert 0\rangle$)	red ($\lvert 0\rangle$)
	110 100 000	0.6312* / 0.6100	0.6710* / 0.7377	0.3289* / 0.2623	red ($\lvert 0\rangle$)	red ($\lvert 0\rangle$)
	101 100 100	0.6996* / 0.7200	0.5821* / 0.5694	0.4179* / 0.4306	red ($\lvert 0\rangle$)	red ($\lvert 0\rangle$)
	110 100 110	0.6470* / 0.6600	0.5229* / 0.5000	0.4771* / 0.5000	none	blue ($\lvert 1\rangle$)
	000 000 110	0.6880* / 0.7000	0.3760* / 0.3714	0.6240* / 0.6286	blue ($\lvert 1\rangle$)	blue ($\lvert 1\rangle$)
	000 100 111	0.5649* / 0.5500	0.2266* / 0.2182	0.7734* / 0.7818	blue ($\lvert 1\rangle$)	blue ($\lvert 1\rangle$)
	100 100 111	0.6236* / 0.6100	0.3330* / 0.3279	0.6670* / 0.6721	blue ($\lvert 1\rangle$)	blue ($\lvert 1\rangle$)
	110 100 101	0.6807* / 0.7500	0.4970* / 0.4933	0.5030* / 0.5067	blue ($\lvert 1\rangle$)	blue ($\lvert 1\rangle$)

Table 5.6: Classification results for input data set II after 100 runs. Trained with training data set II. Theoretical predictions (marked with asterisks) on top, simulation results at the bottom.

slightly favoured due to the distance-dependent weights. Thus, this edge case exposes a slight bias within the training data set II.

In the case of the blue input colours, the first three were all classified correctly with the class qubit probabilities clearly favouring the measurement outcome $\lvert 1\rangle$. Note that the third blue tone was classified correctly despite some red and green content. The last (fourth) blue tone is another edge case with only slightly larger blue than red content. Yet, it was also correctly classified since the probability of measuring class blue is slightly higher.

In conclusion, the quantum simulations of the qubit-based kNN algorithm resulted in three out of four correct classification using training data set I and, ignoring the edge case with equal red and blue content, seven out of seven correct classifications using training dataset II. Hence, with training dataset I the simulated quantum classifier achieved an accuracy of 75%. The accuracy increased to 100% with the use of training dataset II. These results clearly show that the qubit kNN routine, based on distance-weighting with reverse HD, is a very effective classification algorithm with respect to 9-bit RGB colours.

It was shown that for this classification problem the quantum kNN algorithm by Schuld et al. (2014) requires 20 qubits. Unfortunately, the IBM QC consists of only five qubits rendering an actual implementation of the 9-bit RGB colour classification impossible. Furthermore, even when the training and input patterns could be each encoded into only two qubits, the algorithm would require 6 qubits making an IBM QC implementation again impossible. This follows from the initial state $\lvert \psi_0\rangle$ in Equ. 5.4 that contains the input pattern (two qubits), the training pattern (two qubits) as well as one class qubit and one ancilla qubit. This stresses the need for an alternative version of the quantum kNN algorithm based on amplitude-encoded data.

5.2 Development of an amplitude-based kNN algorithm

To enable an implementation of the quantum kNN algorithm using the IBM Quantum Experience platform a new amplitude-based kNN (aKNN) algorithm was developed for this thesis. This algorithm by Schuld, Fingerhuth, and Petruccione (2016) will be introduced in this section using colours for input & training vectors and classes A and B based on the schematic Fig. 2.5 in Section 2.3.1.

The algorithm starts with the assumption that the following initial state can be constructed from M training vectors with N entries:

$$|\psi_0\rangle = \frac{1}{\sqrt{2M}} \sum_{m=1}^{M} (|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^m}\rangle) |c^m(A \ or \ B)\rangle |m\rangle \tag{5.14}$$

where

$$|\Psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle \qquad |\Psi_{t^m}\rangle = \sum_{i=1}^{N} t_i^m |i\rangle \tag{5.15}$$

$$e.g. \quad \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \quad \rightarrow \quad |n\rangle = \sqrt{0.6}|0\rangle + \sqrt{0.4}|1\rangle \tag{5.16}$$

The first qubit in Equ. 5.14 is an ancilla qubit already in an equal superposition of $|0\rangle$ and $|1\rangle$. The ket vector $|\Psi_x\rangle$ which is entangled with the $|0\rangle$ state of the ancilla contains the amplitude-encoded information of the input vector x (red star in Fig. 2.5) as shown in Equ. 5.15. Furthermore, entangled with the $|1\rangle$ state of the ancilla is the ket vector $|\Psi_{t^m}\rangle$ containing the amplitude-encoded information of the training vectors (see also Equ. 5.15). Lastly, there is the class qubit $|c^m(A \ or \ B)\rangle$ and the so-called m-register $|m\rangle$ which is used to separate the training vectors.

Having prepared the initial state $|\psi_0\rangle$ one has to simply apply an H gate to the ancilla qubit. This causes the amplitudes of $|\Psi_x\rangle$ and $|\Psi_{t^m}\rangle$ to interfere. One can think of this like water waves travelling towards each other; constructive interference happens when two crests add up producing a larger wave and destructive interference takes place when a crest and a trough cancel each other out. In this case, constructive $(+)$ interference happens when the ancilla qubit is $|0\rangle$ and destructive $(-)$ interference when the ancilla is $|1\rangle$. The interference of input and training vectors yields the following state,

$$|\psi_1\rangle = (H \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1}) |\psi_0\rangle$$

$$= \frac{1}{2\sqrt{M}} \sum_{m=1}^{M} (|0\rangle [|\Psi_x\rangle + |\Psi_{t^m}\rangle] + |1\rangle [|\Psi_x\rangle - |\Psi_{t^m}\rangle]) |c^m(A \ or \ B)\rangle |m\rangle \tag{5.17}$$

To only select the constructive interference, a CM has to be performed on the ancilla qubit. All previous steps have to be repeated until the ancilla is measured in the $|0\rangle$ state. The probability for this to happen is:

$$Prob(CM) = Prob(|a\rangle = |0\rangle) = 1 - \frac{1}{4M} \sum_{m=1}^{M} \sum_{i=1}^{N} |x_i - t_i^m|^2 \tag{5.18}$$

Note that $\sum_{i=1}^{N} \mid x_i - t_i^m \mid^2$ is the squared Euclidean distance between the input vector and the mth training vector. Equ. 5.18 shows that the probability for the CM to succeed is higher when the average distance between input and training vectors is small. If all training vectors are relatively far from the input the training set can be regarded as suboptimal. Therefore, $Prob(\mid a \rangle = \mid 0 \rangle)$ is a measure of how suitable the training vectors are to classify the new input.

After the successful CM, the state is proportional to:

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^{M} \sum_{i=1}^{N} (x_i + t_i^m) \mid 0 \rangle \mid i \rangle \mid c^m (A \ or \ B) \rangle \mid m \rangle \tag{5.19}$$

The probability of measuring e.g. class $\mid 1 \rangle$ (B) is given by the following expression:

$$Prob(\mid c^m \rangle = \mid 1(B) \rangle) = \sum_{m \mid c^m = 1(B)} 1 - \frac{1}{4M} \sum_{i=1}^{N} \mid x_i - t_i^m \mid^2 \tag{5.20}$$

Note that the probability of measuring class $\mid 1 \rangle$ (B) is dependent on the squared Euclidean distance between the input vector and each training vector belonging to class B. Thus, if the average of these squared Euclidean distances is small, the probability of measuring the class qubit in the $\mid 1 \rangle$ state is greater. This quantum routine, therefore, resembles a kNN algorithm with $k = all$ and without distance-weighting.
All previous steps need to be repeated y times in order to generate a sufficiently accurate picture of the probability distribution of $\mid c \rangle$.

The quantum advantage of the algorithm is the parallel computation of the squared Euclidean distance between the input vector and each training vector through the implementation of a single H gate. Such an operation is impossible to perform on a classical computer. Consider for example a particular training set containing 100,000,000 vectors with 10 entries each: The quantum algorithm performs all 100,000,000 distance computations between input and training vectors within one step whereas a classical computer would need to perform 100,000,000 individual computations to arrive at the same result.

Complexity analysis neglecting initial state preparation

Independent of number and size of the input and training vectors, the algorithm only requires the application of a single H gate. However, the routine has to be repeated for y runs of which only $Prob(CM) * y$ runs result in a measurement of the class qubit. Thus, the time complexity of this algorithm is $\mathcal{O}(\frac{1}{Prob(CM)})$ where $Prob(CM)$ is the probability of a successful CM (measuring ancilla in the $\mid 0 \rangle$ state). Therefore, the algorithm is said to run in constant time. This means it is independent of the number of training vectors and their size. For example, executing this quantum routine with 10 or 100,000,000 training vectors each with 1,000,000 entries does not make a difference in the run time.
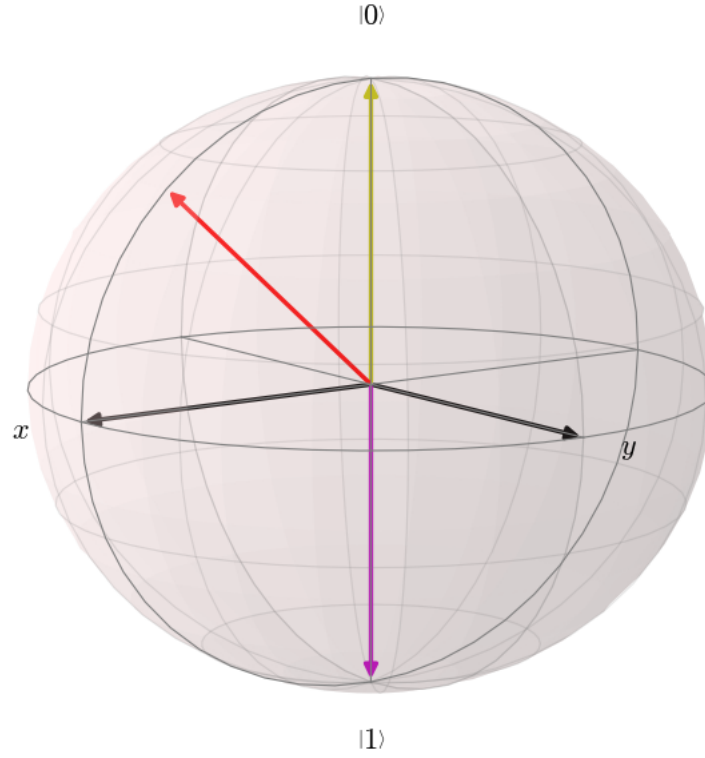
Figure 5.1: Simple binary classification problem of a quantum state

5.2.1 Compiling the amplitude-based kNN algorithm

The akNN algorithm was developed with an actual implementation, using the IBM QC, in mind. Since the IBM Quantum Experience only provides five qubits and a relatively small gate set, a very simple low-dimensional classification problem should be selected.

Perhaps the simplest problem is the classification of a 2-D quantum state vector as either $|0\rangle$ or $|1\rangle$ depending on its position on the Bloch sphere. This choice also enables easy visualization throughout the discussion since any single qubit vector has a well-defined position on the Bloch sphere (see Section 2.1.1). The training set is listed in Table 5.2.1 and consists only of the $|0\rangle$ and the $|1\rangle$ vector depicted in Fig. 5.1 as the yellow and purple vector respectively. The red vector in Fig. 5.1 will be the input vector and its qubit state and vector representation are given in Table 5.2.1.

Qubit state	Vector representation	Class
$|0\rangle$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$|0\rangle$ (yellow)
$|1\rangle$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$|1\rangle$ (purple)

Table 5.7: Training set

Qubit state	Vector representation	Expected class
$e^{-i\frac{\pi}{8}}\left[0.92388\,|0\rangle + 0.38268\,|1\rangle\right]$	$e^{-i\frac{\pi}{8}}\begin{pmatrix} 0.92388 \\ 0.38268 \end{pmatrix}$	$|0\rangle$ (yellow)

Table 5.8: Input vector

Initial state preparation

The first step towards an actual implementation of this problem is to prepare the initial quantum state $|\psi_0\rangle$ previously defined in Equ. 5.14 to be of the form,

$$|\psi_0\rangle = \frac{1}{\sqrt{2M}} \sum_{m=1}^{M} (|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^m}\rangle)) |c^m\rangle |m\rangle \tag{5.21}$$

where in the case of the selected classification problem:

$$|\Psi_x\rangle = e^{-i\frac{\pi}{8}} \left[0.92388 |0\rangle + 0.38268 |1\rangle \right] \tag{5.22}$$

$$|\Psi_{t^1}\rangle = |0\rangle \tag{5.23}$$

$$|\Psi_{t^2}\rangle = |1\rangle \tag{5.24}$$

Since there are only two training vectors the index m in Equ. 5.21 only takes the values 1 and 2. However, the m qubit can only take binary values such that we need to redefine $1 \to 0$ and $2 \to 1$. With this observation, the required number of qubits can be deduced from Equ. 5.21: one ancilla, one qubit for input and training vectors, one class qubit and one m qubit making a total of four qubits. In the subsequent discussion the quantum state of the IBM QC in the ith step will be denoted $|\chi_i\rangle$. Since all qubits in the IBM Quantum Composer are initialized in state $|0\rangle$ the initial state $|\chi_0\rangle$ is simply,

$$|\chi_0\rangle = |a\rangle |d\rangle |c\rangle |m\rangle = |0\rangle |0\rangle |0\rangle |0\rangle \tag{5.25}$$

where a stands for ancilla, d for data, c for class and m for the m qubit. The sum over m is introduced by simply acting an H gate on the m qubit:

$$|\chi_1\rangle = (\mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes H) |0\rangle |0\rangle |0\rangle |0\rangle = \frac{1}{\sqrt{2}} \sum_{m=0}^{1} |0\rangle |0\rangle |0\rangle |m\rangle \tag{5.26}$$

Using another H gate, the ancilla qubit is put in superposition:

$$|\chi_2\rangle = (H \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1}) |\chi_1\rangle = \frac{1}{2} \sum_{m=0}^{1} |0\rangle |0\rangle |0\rangle |m\rangle + |1\rangle |0\rangle |0\rangle |m\rangle = \frac{1}{2} \sum_{m=0}^{1} \left[|0\rangle |0\rangle + |1\rangle |0\rangle \right] |0\rangle |m\rangle \tag{5.27}$$

Next, the input vector $|\Psi_x\rangle$ should be loaded into the quantum state by means of a yet unknown gate sequence GS such that the state is described by,

$$|\chi_3\rangle = GS |\chi_2\rangle = \frac{1}{2} \sum_{m=0}^{1} \left[|0\rangle |\Psi_x\rangle + |1\rangle |0\rangle \right] |0\rangle |m\rangle$$

$$= \frac{1}{2} \sum_{m=0}^{1} \left[|0\rangle \, e^{-i\frac{\pi}{8}} \left[0.92388 |0\rangle + 0.38268 |1\rangle \right] + |1\rangle |0\rangle \right] |0\rangle |m\rangle \tag{5.28}$$

By looking closely at Fig. 5.1 one can deduce that the red input vector can be reached by simply rotating the $|0\rangle$ vector by an angle of $\frac{\pi}{4}$ around the y-axis. A y-rotation by an arbitrary angle ϑ can be achieved with the rotation operator $R_y(\vartheta)$. As described by Nielsen and Chuang (2010), $R_y(\vartheta)$ can be represented as a unitary 2x2 matrix and is obtained from exponentiating the Y gate as shown in Equ. 5.29.

$$R_y(\vartheta) = e^{-i\vartheta \frac{Y}{2}} = \cos\frac{\vartheta}{2}\mathbb{1} - i\sin\frac{\vartheta}{2}Y = \begin{pmatrix} \cos\frac{\vartheta}{2} & -\sin\frac{\vartheta}{2} \\ \sin\frac{\vartheta}{2} & \cos\frac{\vartheta}{2} \end{pmatrix} \tag{5.29}$$

At this point, note that $R_y(\vartheta)$ is not an element of IBM's universal gate set. The problem of how to implement $R_y(\vartheta)$ on the IBM QC will be addressed later. For now, suppose $R_y(\frac{\pi}{4})$ can be implemented. Acting this gate on the data qubit in $|\chi_2\rangle$ yields the expression

$$(\mathbb{1} \otimes R_y(\frac{\pi}{4}) \otimes \mathbb{1} \otimes \mathbb{1})|\chi_2\rangle = \frac{1}{2}\sum_{m=0}^{1}\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_x\rangle\,\big]\,|0\rangle\,|m\rangle \tag{5.30}$$

This, however, is not the desired state $|\chi_3\rangle$ defined in Equ. 5.28 since the input vector $|\Psi_x\rangle$ should only be entangled with the $|0\rangle$ state of the ancilla. To achieve this type of entanglement the controlled version of the y-rotation gate, $CR_y(\frac{\pi}{4})(c,t)$, is required. When applying it using the a qubit as control and the d qubit as target the input vector will be entangled with the $|1\rangle$ state of the ancilla. Flipping the a qubit with an X gate moves the input vector to the $|0\rangle$ state of the ancilla. Hence, the desired state $|\chi_3\rangle$ is obtained by applying the following gate sequence GS:

$$GS = (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})(CR_y(\frac{\pi}{4})(a,d) \otimes \mathbb{1} \otimes \mathbb{1}) \tag{5.31}$$

subbing into Equ. 5.28:

$$|\chi_3\rangle = GS\,|\chi_2\rangle = (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})(CR_y(\frac{\pi}{4})(a,d) \otimes \mathbb{1} \otimes \mathbb{1})|\chi_2\rangle$$

$$= (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})\Big[\frac{1}{2}\sum_{m=0}^{1}\big[\,|0\rangle\,|0\rangle + |1\rangle\,|\Psi_x\rangle\,\big]\,|0\rangle\,|m\rangle\Big]$$

$$= \frac{1}{2}\sum_{m=0}^{1}\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|0\rangle\,\big]\,|0\rangle\,|m\rangle \tag{5.32}$$

It is important to note that also $CR_y(\frac{\pi}{4})(c,t)$ is not an element of IBM's universal gate set. Its implementation on the IBM QC will be discussed in the next subsection.

The next step is to entangle the first training vector $|\Psi_{t^0}\rangle$ with the $|1\rangle$ state of the ancilla and the $|0\rangle$ state of the m qubit. Additionally, the second training vector $|\Psi_{t^1}\rangle$ should be entangled with the $|1\rangle$ states of the ancilla and the m qubit. Note that $|\Psi_{t^1}\rangle$ and $|\Psi_{t^2}\rangle$ defined in Equ. 5.22 were redefined to $|\Psi_{t^0}\rangle$ and $|\Psi_{t^1}\rangle$ respectively. Expanding the sum in Equ. 5.32 demonstrates that $|\Psi_{t^0}\rangle = |0\rangle$ is already at its desired place:

$$|\chi_3\rangle = \frac{1}{2}\Big[\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|0\rangle\,\big]\,|0\rangle\,|0\rangle + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|0\rangle\,\big]\,|0\rangle\,|1\rangle\Big]$$

$$= \frac{1}{2}\Big[\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big]\,|0\rangle\,|0\rangle + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|0\rangle\,\big]\,|0\rangle\,|1\rangle\Big] \tag{5.33}$$

In order to entangle $|\Psi_{t^1}\rangle$ with the $|1\rangle$ states of the ancilla and m qubit a Toffoli (CCNOT) gate needs to be implemented. Using the ancilla (a) and m qubit as controls and choosing the data (d) qubit as target one obtains the following state:

$$|\chi_4\rangle = CCNOT(a,m,d)\,|\chi_3\rangle$$

$$= \frac{1}{2}\Big[\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big]\,|0\rangle\,|0\rangle + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|1\rangle\,\big]\,|0\rangle\,|1\rangle\Big]$$

$$= \frac{1}{2}\Big[\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big]\,|0\rangle\,|0\rangle + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^1}\rangle\,\big]\,|0\rangle\,|1\rangle\Big] \tag{5.34}$$

The class qubit for the first training vector is already in the correct $|0\rangle$ state. Note again that also the CCNOT gate is not an element of IBM's universal gate set which will be addressed later.

It remains to flip the class qubit for the second training vector by applying a CNOT gate using the d qubit as control and the class(c) qubit as target. The resulting state is then given by,

$$
\begin{aligned}
|\chi_5\rangle &= CNOT(d,c)\,|\chi_4\rangle \\
&= \frac{1}{2}\Big[\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big]\,|0\rangle\,|0\rangle + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^1}\rangle\,\big]\,|1\rangle\,|1\rangle\,\Big]
\end{aligned}
\tag{5.35}
$$

and can be rewritten as:

$$
\begin{aligned}
|\chi_5\rangle &= \frac{1}{2}\sum_{m=1}^{2}\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big] + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^1}\rangle\,\big]\,|c^m\rangle\,|m\rangle \\
&= \frac{1}{2}\sum_{m=1}^{2}\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^m}\rangle\,\big]\,|c^m\rangle\,|m\rangle
\end{aligned}
\tag{5.36}
$$

When comparing Equ. 5.36 to $|\psi_0\rangle$ in Equ. 5.14 it becomes clear that $|\chi_5\rangle$ is in the form of the desired initial quantum state $|\psi_0\rangle$. The quantum state preparation is therefore theoretically completed.

Decomposition of a controlled-U gate

To implement the quantum state preparation on the IBM QC, it remains to find a way of realizing the controlled y-rotation $CR_y(\frac{\pi}{4})(c,t)$ using only the ten gates from IBM's universal gate set. In their book Nielsen and Chuang (2010) describe how a controlled U (CU) gate can be decomposed into a sequence of CNOT and single qubit gates. Thereby, U can be any unitary single-qubit gate. A CU gate is then defined as:

$$
CU = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & U \end{pmatrix}
\tag{5.37}
$$

Most of the time the CU gate cannot be implemented directly since it is not element of the universal gate set at hand and it has to be realized through larger quantum circuits. Fig. 5.2 shows the decomposition described by Nielsen and Chuang (2010) into two CNOTs, three unitary single-qubit gates A, B, C and a phase-adjusting matrix which will be denoted P of the form:

$$
P = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}
\tag{5.38}
$$

The idea of this decomposition is that when the control qubit (top qubit in Fig. 5.2) is $|0\rangle$ the gate combination ABC is applied to the target qubit (bottom qubit in Fig. 5.2) and has to equal the identity gate:

$$
ABC = \mathbb{1}
\tag{5.39}
$$

---

[14]Reprinted from Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000. Copyright 2010 by Nielsen & Chuang.
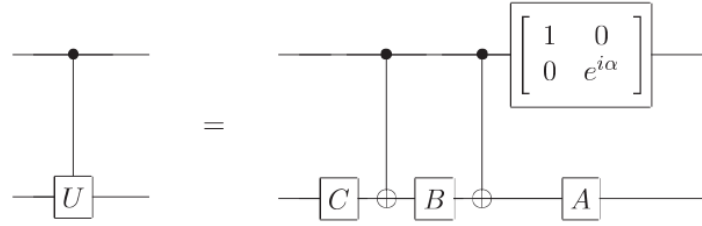
Figure 5.2: Circuit decomposition of controlled-U quantum gate.[14]

If and only if the control qubit is $|1\rangle$ then the gate sequence $e^{i\alpha}AXBXC$ is applied to the target. Since the goal is to apply the unitary U to the target qubit the following equation must be satisfied:

$$e^{i\alpha}AXBXC = U \tag{5.40}$$

Nielsen and Chuang (2010) make the following choices for the unitary gates A, B, C:

$$A = R_z(\beta)R_y(\frac{\gamma}{2}) \tag{5.41}$$

$$B = R_y(-\frac{\gamma}{2})R_z(-\frac{\delta + \beta}{2}) \tag{5.42}$$

$$C = R_z(\frac{\delta - \beta}{2}) \tag{5.43}$$

where $R_z(\varrho)$ is the general rotation gate about the z-axis of the Bloch sphere. Similarly to the $R_y(\vartheta)$ gate it can be obtained by exponentiating the Z gate as shown below in Equ. 5.44.

$$R_z(\varrho) = e^{-i\varrho\frac{Z}{2}} = \cos\frac{\varrho}{2}\mathbb{1} - i\sin\frac{\varrho}{2}Z = \begin{pmatrix} e^{-i\frac{\varrho}{2}} & 0 \\ 0 & e^{i\frac{\varrho}{2}} \end{pmatrix} \tag{5.44}$$

When subbing the expressions for A, B, C from Equ. 5.41 into Equ. 5.39 one will indeed obtain the identity operation (proof omitted). These choices of A, B, C are also a solution to Equ. 5.40. Subbing into Equ. 5.40 leads to the following expression for matrix U:

$$U = \begin{pmatrix} e^{i(\alpha - \frac{\beta}{2} - \frac{\delta}{2})}\cos\frac{\gamma}{2} & -e^{i(\alpha - \frac{\beta}{2} + \frac{\delta}{2})}\sin\frac{\gamma}{2} \\ e^{i(\alpha + \frac{\beta}{2} - \frac{\delta}{2})}\sin\frac{\gamma}{2} & e^{i(\alpha + \frac{\beta}{2} + \frac{\delta}{2})}\cos\frac{\gamma}{2} \end{pmatrix} \tag{5.45}$$

To decompose $CR_y(\frac{\pi}{4})$, one simply chooses $U = R_y(\frac{\pi}{4})$. Subbing $\vartheta = \frac{\pi}{4}$ into Equ. 5.29 the matrix representation of $R_y(\frac{\pi}{4})$ is obtained:

$$U = R_y(\frac{\pi}{4}) = \begin{pmatrix} \cos\frac{\pi}{8} & -\sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & \cos\frac{\pi}{8} \end{pmatrix} \tag{5.46}$$

Subbing this expression for U into Equ. 5.45 leads to:

$$U = \begin{pmatrix} \cos\frac{\pi}{8} & -\sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & \cos\frac{\pi}{8} \end{pmatrix} = \begin{pmatrix} e^{i(\alpha - \frac{\beta}{2} - \frac{\delta}{2})}\cos\frac{\gamma}{2} & -e^{i(\alpha - \frac{\beta}{2} + \frac{\delta}{2})}\sin\frac{\gamma}{2} \\ e^{i(\alpha + \frac{\beta}{2} - \frac{\delta}{2})}\sin\frac{\gamma}{2} & e^{i(\alpha + \frac{\beta}{2} + \frac{\delta}{2})}\cos\frac{\gamma}{2} \end{pmatrix} \tag{5.47}$$

When setting the expressions for the respective matrix entries in Equ. 5.47 equal, the following system of non-linear complex equations is obtained :

$$\cos\frac{\pi}{8} = e^{i(\alpha - \frac{\beta}{2} - \frac{\delta}{2})}\cos\frac{\gamma}{2} \tag{5.48}$$

$$-\sin\frac{\pi}{8} = -e^{i(\alpha - \frac{\beta}{2} + \frac{\delta}{2})}\sin\frac{\gamma}{2} \tag{5.49}$$

$$\sin\frac{\pi}{8} = e^{i(\alpha + \frac{\beta}{2} - \frac{\delta}{2})}\sin\frac{\gamma}{2} \tag{5.50}$$

$$\cos\frac{\pi}{8} = e^{i(\alpha + \frac{\beta}{2} + \frac{\delta}{2})}\cos\frac{\gamma}{2} \tag{5.51}$$

$$\tag{5.52}$$

This is a system of four non-linear complex equations with four unknowns. In order to solve for the parameters α, β, γ and δ one can use any root finding algorithm for non-linear equations such as Secant or Newton's method. Using Newton's method the solutions are found to be:

$$\alpha = \pi; \quad \beta = 2\pi; \quad \delta = \frac{7}{8}\pi; \quad \gamma = 0 \tag{5.53}$$

Subbing these parameters into the expressions for A, B and C defined in Equ. 5.41 yields,

$$A = R_z(\beta)R_y(\frac{\gamma}{2}) = R_z(2\pi) = \mathbb{1} \tag{5.54}$$

$$B = R_y(-\frac{\gamma}{2})R_z(-\frac{\delta + \beta}{2}) = R_z(-\frac{23}{16}\pi) = \text{?} \tag{5.55}$$

$$C = R_z(\frac{\delta - \beta}{2}) = R_z(-\frac{9}{16}\pi) = \text{?} \tag{5.56}$$

$$P = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} = Z \tag{5.57}$$

Quantum gate A is just equal to the identity gate which certainly is an element of IBM's universal gate set. The phase-adjusting gate P is equal to the Z gate that also is within IBM's gate set. Only gates B and C result in more complex z-rotations of $-\frac{23}{16}\pi$ and $-\frac{9}{16}\pi$ radians respectively. Unfortunately, these quantum gates are not found in IBM's gate set as indicated by the red question marks in Equ. 5.55 and 5.56.

The Solovay-Kitaev theorem

An important theorem in quantum information is the Solovay-Kitaev theorem first proposed by Robert M. Solovay and later published by Kitaev (1995). It will be used to decompose the quantum gates B and C into a sequence of quantum gates from IBM's universal gate set.

At this point, it is important to remember that any universal quantum gate set is a dense subset of the special unitary group $SU(2)$ as previously defined in Section 2.1.1.

Theorem: Solovay-Kitaev theorem

Let G be a universal quantum set G consisting of quantum gates from $SU(2)$ and let $\epsilon > 0$ be a desired accuracy. Then there is a constant b such that for any single-qubit gate $W \in SU(2)$ there exists a finite gate sequence $\tilde{G}$ of gates from the set G of length $O(log^b(1/\epsilon))$ such that $d(W, \tilde{G}) < \epsilon$. In their proof, Dawson and Nielsen (2005) show that $b \approx 3.97$.

In other words, given a set of single-qubit quantum gates which is a dense subset of $SU(2)$, then the Solovay-Kitaev theorem guarantees that this set will quickly fill $SU(2)$. (Dawson & Nielsen, 2005).

The Solovay-Kitaev theorem is important because it proofs that given any universal gate set it is possible to obtain good approximations to any desired single-qubit gate W. In their paper, Dawson and Nielsen (2005) describe how to design an algorithm implementing the Solovay-Kitaev theorem. Unfortunately, the Solovay-Kitaev algorithm needs to be implemented on a classical computer which adds to the overall time complexity of the quantum compiling process as will be shown later. For this thesis, the open-source software package 'Quantum Compiler, v0.03'[15] developed in Python by Paul Pham was used to run the Solovay-Kitaev algorithm described by Dawson and Nielsen (2005). This software package compares different gate approximations to the desired gate by a new distance metric called Fowler distance.

Definition: Fowler distance

When approximating a unitary gate W with a gate sequence yielding another unitary gate W_{approx} it is important to quantify how 'close' the action of W_{approx} is to W. The Fowler distance makes use of the matrix representations of W and W_{approx} and is defined by Booth Jr (2012) as:

$$d(W, W_{approx}) = \sqrt{\frac{2 - \mid tr(W \cdot W_{\text{approx}}^{\dagger}) \mid}{2}} \qquad (5.58)$$

In contrast to W, W_{approx} might introduce a shift in the global phase of the quantum state it is acting on. However, since the global phase of a quantum state is immeasurable in experiment it can be neglected. As a result, the Fowler distance is defined in such a way that possible global phase differences are ignored.

Subsequently, the desired gates $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$ where decomposed using the Solovay-Kitaev algorithm. Fig. 5.3 shows a plot visualizing how the Fowler distance depends on the number of gates in the approximating gate sequence W_{approx}. The number of gates in the approximating gate sequence is also called *gate count*. The plot clearly shows an exponential increase in the gate count for decreasing Fowler distance. For example, 146 gates suffice to achieve $d(W_{approx}, R_z(-\frac{9}{16}\pi)) = 0.04389$. Yet, already 17,838 gates are required to reduce the Fowler distance to $d(W_{approx}, R_z(-\frac{9}{16}\pi)) = 0.01008$.

---

[15]The software package 'Quantum Compiler, v0.03' by Paul Pham can be downloaded from `https://sourceforge.net/projects/quantumcompiler/files/v0.03/`.
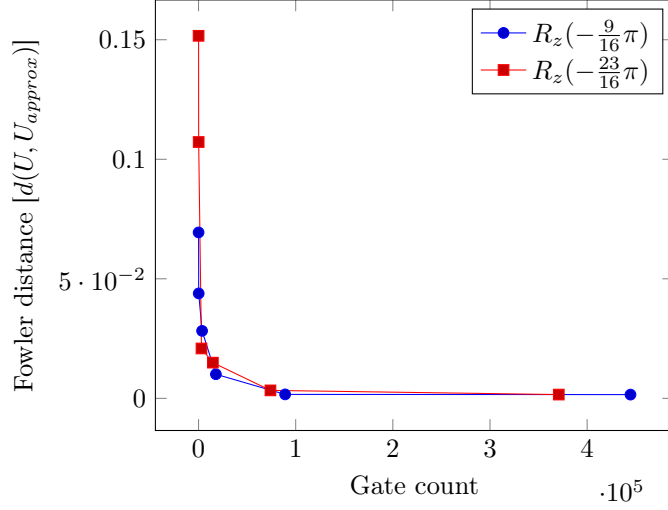
Figure 5.3: Fowler distance plotted against the required gate count for quantum gates $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$

Fowler distance is best understood with a visual example using the gate $B = R_z(-\frac{23}{16}\pi)$. Acting the desired gate B on the state $|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ (black vector in Fig. 5.4) results in the purple Bloch vector $\vec{b}$ shown in Fig. 5.4.
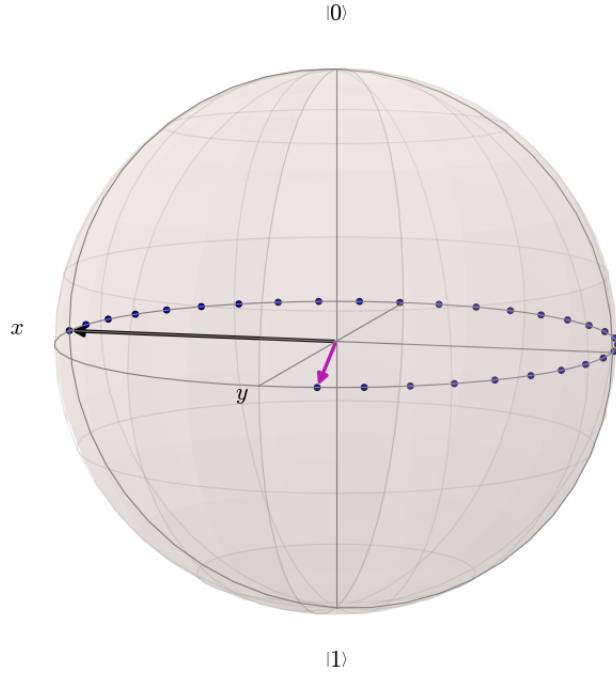
Figure 5.4:   Action of $B = R_z(-\frac{23}{16}\pi)$ on the state $|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$

Fig. 5.5 shows the action of four different approximating gate sequences $B_{approx,1}$, $B_{approx,2}$, $B_{approx,3}$, $B_{approx,4}$ on the state $|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$. Gate sequence $B_{approx,1}$ consists of 25 gates and is a Fowler distance of $d = 0.15165$ away from B. The resulting state vector is coloured green in Fig. 5.5. The green vector lies above the Bloch equator and is relatively far away from the desired

vector $\vec{b}$ in Fig. 5.4. With 109 gates, $B_{approx,2}$ yields a Fowler distance of $d = 0.10722$ resulting in the blue vector. It is also found below the Bloch equator and still relatively far from $\vec{b}$. Using $B_{approx,3}$ the Fowler distance drops to $d = 0.02086$ leading to the yellow vector in Fig. 5.5. This vector is almost on the Bloch equator and relatively close to $\vec{b}$ implying that $B_{approx,3}$ is a good approximation of B. However, $B_{approx,3}$ already requires the implementation of 2,997 single-qubit gates. The purple vector from Fig. 5.4 is also plotted in Fig. 5.5 but is not visible since the red vector with $d = 0.00158$ constitutes a very good approximation to the desired state $\vec{b}$. The red vector is the result of the action of $B_{approx,4}$ which, unfortunately, needs 370,813 gates to achieve such a small Fowler distance. Hereby, Gate B was used as an example but such an exponential increase in the gate count is observed for any quantum gate W when applying the Solovay-Kitaev algorithm (Dawson & Nielsen, 2005).

$|0\rangle$

x

y

$d = 0.15165$

$d = 0.10722$

$d = 0.02086$

$d = 0.00158$

$|1\rangle$
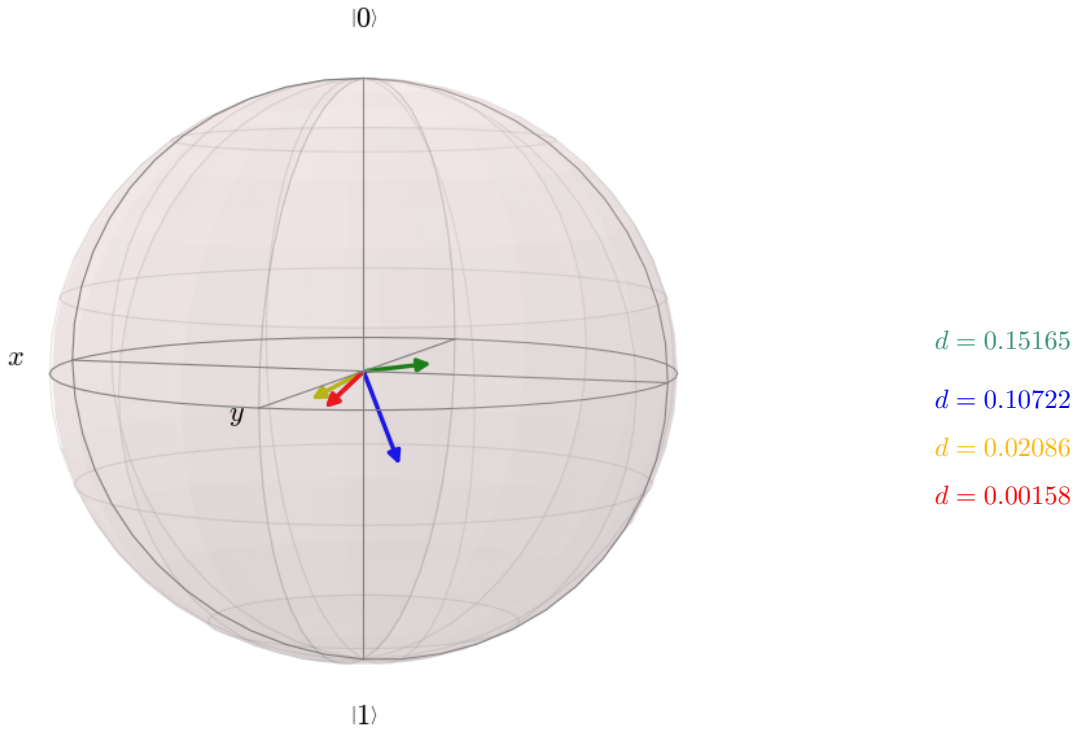
Figure 5.5: Various Fowler distances visualized on Bloch sphere

Implementing any quantum gate D inevitably takes some time t since it involves the use of hardware components e.g. targeting a specific electron with a laser pulse. Based on IBM's single-qubit and CNOT gate times described in Section 3.2 the total execution time of twelve different gate sequences, approximating $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$ to different Fowler distances, were calculated. The results can be seen in Table 5.9. Keeping in mind that in the best case, the maximum decoherence time of a qubit in IBM's QC is $112.4\,\mu s$ most gate sequences are too long for an IBM implementation (coloured red in Table 5.9). Feasible execution times are marked green in Table 5.9. However, decoherence is not the only limiting factor since IBM only allows for 39 gates and one measurement gate. But even when selecting the smallest sequences of 25 gates for B and 16 gates for C, the total gate count is 41 which is two gates more than the allowed gate count of the IBM Quantum Composer. Furthermore, the approximating gate sequences of B and C are not the only gates needed to prepare the initial quantum state and execute the akNN algorithm. Unfortunately, this makes an actual implementation of this particular classification problem on IBM's QC impossible.

Approx. Gate	Fowler distance	Gate count	Execution time
$R_z(-\frac{23}{16}\pi)$	0.15165	25	$\sim3\,\mu s$
	0.10722	109	$\sim14\,\mu s$
	0.02086	2,997	$\sim390\,\mu s$
	0.01494	14,721	$\sim1914\,\mu s$
	0.003327	74,009	$\sim9621\,\mu s$
	0.001578	370,813	$\sim48\,206\,\mu s$
$R_z(-\frac{9}{16}\pi)$	0.28390	16	$\sim2\,\mu s$
	0.04389	146	$\sim18\,\mu s$
	0.049511	728	$\sim87\,\mu s$
	0.02823	3,622	$\sim435\,\mu s$
	0.01008	17,838	$\sim2141\,\mu s$
	0.00156	444,646	$\sim53\,358\,\mu s$

Table 5.9: Results of the Solovay-Kitaev algorithm for decomposition of $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$

To complete the quantum compiling, assume that gates B and C could somehow be implemented with shorter gate sequences. It remains to decompose the Toffoli gate that was required to 'load' the second training pattern $|\Psi_{t^1}\rangle$ into the quantum state $|\chi_4\rangle$ as shown in Equ. 5.34.

Toffoli Gate Decomposition

The Toffoli or CCNOT gate can be decomposed into CNOT and single-qubit gates in several ways. In their book, Nielsen and Chuang (2010) describe the decomposition into six CNOTs and nine single-qubit gates; two H gates, three T and three T† gates. See Fig. 5.7 for the corresponding circuit diagram. This decomposition does not require any further compiling work since all involved gates are elements of IBM's universal gate set.

Figure 5.6: Decomposition of a Toffoli gate into six CNOT and nine single-qubit gates[16]

The Toffoli decomposition finalizes the compiling of the quantum state preparation into gates from IBM's gate set only. In summary, the compiled state preparation requires: 2 H gates applied to the ancilla and the m qubit, the decomposed $CR_y(\frac{\pi}{4})$ gate (2 CNOTs, 1 Z gate, 1 $\mathbb{1}$ gate (can be neglected) & a sequence of minimum 25 gates for B and minimum 16 gates for C), 1 X gate to flip the ancilla (Equ. 5.32), 1 decomposed Toffoli (6 CNOTs, 2 H gates, 3 T & 3 T† gates) to load the second training vector and 1 CNOT to flip the class qubit. Thus, the quantum state preparation was compiled into 9 CNOT and 53 single-qubit gates. The akNN algorithm only requires one additional H gate and two measurement gates (for ancilla and class qubit) leading to a total of 9 CNOT, 54 single-qubit gates and 2 measurement gates. However, even cleverly arranging these gates does not suffice to implement the algorithm within IBM's 40 gate slots. The full circuit diagram for the compiled state preparation and the akNN algorithm can be seen in Fig. 5.7.

---

[16]Reprinted from Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000. Copyright 2010 by Nielsen & Chuang.

Figure 5.7: Compiled quantum circuit for implementing the akNN algorithm with the binary Bloch vector classification problem

Complexity analysis including quantum compiling steps

The akNN algorithm itself was found to run in constant time $\mathcal{O}(\frac{1}{Prob(CM)})$. However, to determine the overall algorithmic complexity all steps required to prepare the initial quantum state $|\psi_0\rangle$ from Equ. 5.14 need to be included. The decomposition of the $CR_y(\frac{\pi}{4})$ gate involved solving a system of non-linear equations by means of a root-finding algorithm. Any iterative root-finding algorithm e.g. Secant or Newton's method has a complexity of $\mathcal{O}(k)$ where k is the number of root finding iterations. Furthermore, the Solovay-Kitaev algorithm was required to find two single-qubit gate sequences approximating the two gates $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac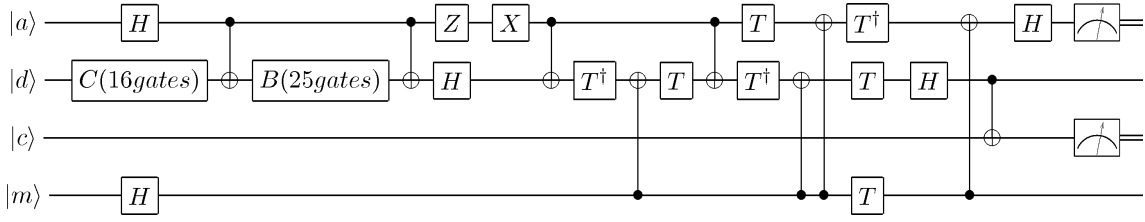{9}{16}\pi)$. According to Dawson and Nielsen (2005) the Solovay-Kitaev algorithm has a complexity of $\mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon}))$ for ϵ-approximations of m gates. Thus the total complexity is given by:

$$\mathcal{O}(\frac{1}{Prob(CM)}) + \mathcal{O}(k) + \mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon})) \tag{5.59}$$

When adding algorithmic complexities only the most dominant term is considered relevant. Thus, the overall complexity including state preparation is found to be $\mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon}))$.

In conclusion, due to the necessary quantum compiling steps the initial constant complexity of the akNN algorithm

$$\mathcal{O}(\frac{1}{Prob(CM)}) \tag{5.60}$$

increased to a polylogarithmic complexity of

$$\mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon})) \tag{5.61}$$

dependening on the number of gates m that need ϵ-approximations by means of the Solovay-Kitaev algorithm.

5.2.2 Simulating the amplitude-based kNN algorithm

Since the IBM Quantum Experience does not allow for an implementation, the aKNN algorithm was simulated in Liqui$|\rangle$ to determine the performance and the outcome of the algorithm. This section is subdivided into two parts: First, the classification of Bloch vectors described in the last section is simulated. Second, the aKNN algorithm is used to classify different Gaussian distributions constituting a slightly more complex classification task.

Classification of Bloch vectors

The Bloch vector classification problem described in Section 5.2.1 was reconsidered and simulated in Liqui|⟩ to determine if the algorithm yields the expected classification outcome.

Section 5.2.1 pointed out that the implementation of the controlled y-rotation $CR_y(\frac{\pi}{4})$ was the main issue preventing an implementation within IBM's 40 gate slots. However, by specifying its matrix representation Liqui|⟩ allows any unitary single- or multi-qubit gate to be defined within its programming framework. Therefore, $CR_y(\frac{\pi}{4})$ can be easily implemented in Liqui|⟩. This enables to load the input vector specified in Table 5.2.1 without having to decompose the controlled y-rotation gate and with no need for the Solovay-Kitaev algorithm. Due to this enormous simplification of the quantum state preparation routine, a second input vector was considered for simulation. Fig. 5.8 shows the new second input vector on the Bloch sphere. This new input vector can be obtained by rotating the $|0\rangle$ state by $\frac{\pi}{8}$ radians around the y-axis. The resulting new input data set is listed in Table 5.10.

Table 5.10 shows that the probability of measuring the new input vector (ID=2) in state $|0\rangle$ is slightly higher than for the input vector with ID=1. This can also be seen visually when comparing Fig. 5.8 to Fig. 5.1. Lastly, the training data set listed in Table 5.2.1 was not altered for simulation.

ID	Qubit state	Vector representation	Expected class
1	$e^{-i\frac{\pi}{8}}\left[0.92388\,|0\rangle+ 0.38268\,|1\rangle\right]$	$e^{-i\frac{\pi}{8}}\begin{pmatrix}0.92388\\0.38268\end{pmatrix}$	$|0\rangle$ (yellow)
2	$e^{-i\frac{\pi}{16}}\left[0.98079\,|0\rangle+ 0.19509\,|1\rangle\right]$	$e^{-i\frac{\pi}{16}}\begin{pmatrix}0.98079\\0.19509\end{pmatrix}$	$|0\rangle$ (yellow)

Table 5.10: Input data set II for the simulation of the aKNN algorithm. The data set consists of two different qubit states both lying in the z-x plane of the Bloch sphere.
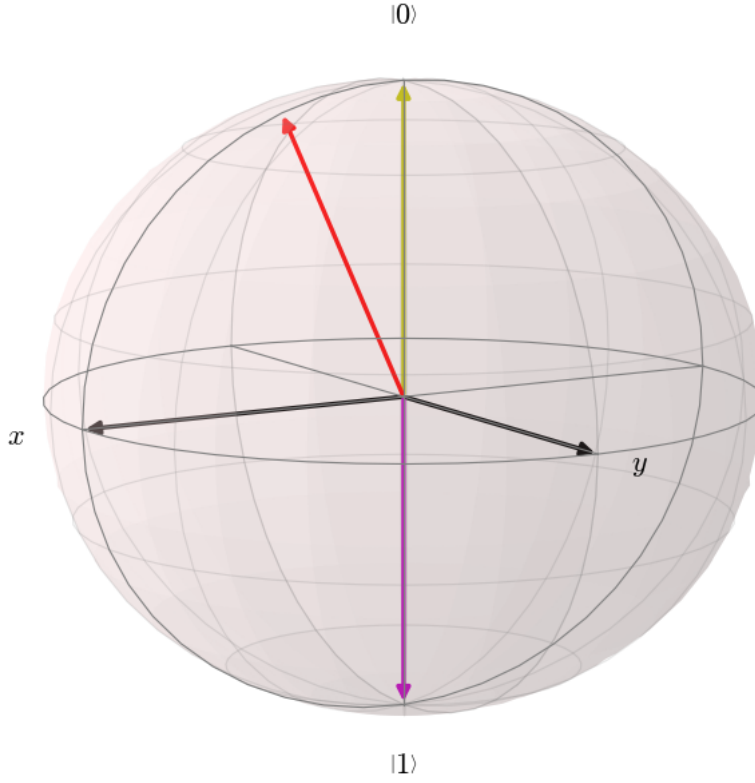
Figure 5.8: Visualization of the second input qubit state $e^{-i\frac{\pi}{16}}\left[0.98079\left|0\right\rangle + 0.19509\left|1\right\rangle\right]$ coloured red on the Bloch sphere. The first training vector $\left|0\right\rangle$ is shown in yellow and the second training vector $\left|1\right\rangle$ is shown in purple.

The obtained simulation results after 1000 runs for both Bloch vectors from input data set II are shown in Table 5.11. The theoretically predicted probabilities are marked with asterisks and are always displayed on top of the corresponding simulation result. Comparing these values shows strong agreement between prediction and simulation implying that 1000 simulation runs suffice to retrieve the expected probability distributions.

ID	Qubit state	$Prob(CM)$	$Prob$ $(\left|c\right\rangle = \left|0\right\rangle)$	$Prob$ $(\left|c\right\rangle = \left|1\right\rangle)$	Expected class	Algorithm output
1	$e^{-i\frac{\pi}{8}}\left[0.92388\left|0\right\rangle + 0.38268\left|1\right\rangle\right]$	0.8266* ——— 0.8050	0.5818* ——— 0.5863	0.4182* ——— 0.4137	$\left|0\right\rangle$ (yellow)	$\left|0\right\rangle$ (yellow)
2	$e^{-i\frac{\pi}{16}}\left[0.98079\left|0\right\rangle + 0.19509\left|1\right\rangle\right]$	0.7940* ——— 0.7710	0.6237* ——— 0.6420	0.3763* ——— 0.3580	$\left|0\right\rangle$ (yellow)	$\left|0\right\rangle$ (yellow)

Table 5.11: Classification results for two qubit states after 1000 runs. Input state with ID=1 was defined in Table 5.2.1 and qubit state with ID=2 was defined in Section 5.2.2. Trained with training set defined in Table 5.2.1 consisting of the $\left|0\right\rangle$ and $\left|1\right\rangle$ qubit states. Theoretical predictions (marked with asterisks) on top, simulation results at the bottom.

Both input qubit states were correctly classified as $\left|0\right\rangle$ (yellow vector in Fig. **??** and Fig. **??**) as expected since both qubit states lie within the upper half of the Bloch sphere. The input qubit

state with ID=2 has a slightly higher probability of being classified as $|0\rangle$ than the qubit state with ID=1. This is expected since the state with ID=2 is closer to the training state $|0\rangle$ on the Bloch sphere than the state with ID=1.

In conclusion, using the small input data set II the amplitude-based kNN algorithm has achieved 100% accuracy. However, since only two Bloch vectors were considered this number should not be taken too seriously but rather be seen as a small test bench for the amplitude-based kNN algorithm. All in all, the main purpose of this simulation was to show that the amplitude-based kNN algorithm would have performed as expected when an implementation with IBM's quantum computer would have been feasible.

The classification of Bloch vectors considered in this section only made use of one qubit for each training and input vector. Since one qubit can only be in a superposition of maximally two states (e.g. the $|0\rangle$ and $|1\rangle$ state) only two amplitudes, e.g. one for the $|0\rangle$ and one for the $|1\rangle$ state, are available for data encoding. This was deliberately chosen since it made it relatively easy to construct the initial quantum state (Equ. 5.14) required for the amplitude-based kNN algorithm. The next section will increase the number of qubits used to encode each training and input vector and will describe how the simulated amplitude-based kNN algorithm can be used to classify different Gaussian distributions.

Classification of Gaussian distributions

Section 4.1.2 introduced the idea of encoding classical data into the 2^n amplitudes of a quantum state with n qubits leading to exponential data compression compared to classical computers. Initializing arbitrary amplitude distributions is still actively researched and as discussed in Section 4.1.2 requires the use of non-trivial quantum algorithms. Yet, this section will demonstrate and use the fact that it is relatively easy to initialize Gaussian amplitude distributions. Thereafter, the amplitude-based kNN algorithm will be used to classify different Gaussian distributions in Liqui$|\rangle$. However, a few important concepts, heavily used in the later discussion, need to be introduced first.

Consider the following classical probability vector w with eight entries representing a discrete Gaussian distribution:

$$w = \begin{pmatrix} 0.58566 \\ 0.11434 \\ 0.11434 \\ 0.022322 \\ 0.11434 \\ 0.022322 \\ 0.022322 \\ 0.0043579 \end{pmatrix} \tag{5.62}$$

The goal is to initialize an amplitude distribution of a multi-qubit system such that it represents the classical vector w. Since w has eight entries the multi-qubit system needs to consist of three qubits giving rise to $2^3 = 8$ amplitudes. Thus, the desired quantum memory state $|w\rangle$ representing the classical vector w should be of the form:

$$\begin{aligned} |w\rangle = \quad & 0.58566\,|000\rangle + 0.11434\,|001\rangle + 0.11434\,|010\rangle + 0.02232\,|011\rangle \\ & + 0.11434\,|100\rangle + 0.02232\,|101\rangle + +0.02232\,|110\rangle + 0.00436\,|111\rangle \end{aligned} \tag{5.63}$$

Knowing that the classical vector w is Gaussian-distributed it follows that the amplitudes of $|w\rangle$ are also Gaussian-distributed which will become important later. To initialize the state $|w\rangle$ a suitable

quantum gate is required. For this purpose, the idea of a coin operator can be borrowed from the theory of quantum random walks (CITATION). Such a coin operator can be used as a quantum gate to initialize a Gaussian distribution centered around a chosen binary qubit pattern as will be shown later. For this purpose, the coin gate, denoted C, will be defined as the following unitary matrix:

$$C(\delta) = \begin{pmatrix} \sqrt{\delta} & 1 - \sqrt{\delta} \\ 1 - \sqrt{\delta} & -\sqrt{\delta} \end{pmatrix} \tag{5.64}$$

where $0.5 \leq \delta \leq 1$. The action of $C(\delta)$ on a quantum state needs to be analyzed to understand how it can be used to create Gaussian-distributed quantum states and how the parameter δ influences the shape of these distributions. For example, acting $C(\delta)$ on each individual qubit in the three-qubit state $|000\rangle$ yields:

$$
\begin{aligned}
(C(\delta) \otimes C(\delta) \otimes C(\delta)) |000\rangle &= C(\delta) |0\rangle \otimes C(\delta) |0\rangle \otimes C(\delta) |0\rangle \\
&\equiv \begin{pmatrix} \sqrt{\delta} & 1 - \sqrt{\delta} \\ 1 - \sqrt{\delta} & -\sqrt{\delta} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} \sqrt{\delta} & 1 - \sqrt{\delta} \\ 1 - \sqrt{\delta} & -\sqrt{\delta} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} \sqrt{\delta} & 1 - \sqrt{\delta} \\ 1 - \sqrt{\delta} & -\sqrt{\delta} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \otimes \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \otimes \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{\delta} * \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \\ (1 - \sqrt{\delta}) * \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \end{pmatrix} \otimes \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \\
&= \begin{pmatrix} \delta \\ \sqrt{\delta} - \delta \\ \sqrt{\delta} - \delta \\ 1 - 2\sqrt{\delta} + \delta \end{pmatrix} \otimes \begin{pmatrix} \sqrt{\delta} \\ 1 - \sqrt{\delta} \end{pmatrix} \\
&= \begin{pmatrix} \delta\sqrt{\delta} \\ \delta - \delta\sqrt{\delta} \\ \delta - \delta\sqrt{\delta} \\ \sqrt{\delta}(\delta + 1) - 2\delta \\ \delta - \delta\sqrt{\delta} \\ \sqrt{\delta}(\delta + 1) - 2\delta \\ \sqrt{\delta}(\delta + 1) - 2\delta \\ -\sqrt{\delta}(3 + \delta) + 1 + 3\delta \end{pmatrix}
\end{aligned} \tag{5.65}
$$

When choosing $\delta = 0.7$, the last expression in Equ. 5.65 is equal to the desired quantum memory state $|w\rangle$ defined in Equ. 5.63:

$$|w\rangle = \begin{pmatrix} 0.58566 \\ 0.11434 \\ 0.11434 \\ 0.022322 \\ 0.11434 \\ 0.022322 \\ 0.022322 \\ 0.0043579 \end{pmatrix}$$

$$\equiv 0.58566 \, |000\rangle + 0.11434 \, |001\rangle + 0.11434 \, |010\rangle + 0.02232 \, |011\rangle$$
$$+ \, 0.11434 \, |100\rangle + 0.02232 \, |101\rangle + +0.02232 \, |110\rangle + 0.00436 \, |111\rangle \tag{5.66}$$

Thus, it was demonstrated that acting $C(0.7)$ on each qubit in the state $|000\rangle$ yields the desired quantum memory state $|w\rangle$. Since the amplitudes of $|w\rangle$ represent a discretized Gaussian distribution, it was shown that the coin gate $C(0.7)$ can indeed be used to generate Gaussian amplitude distributions. As will be shown later, for any δ the gate $C(\delta)$ creates a different Gaussian amplitude distribution. Next, two different ways of visualising these Gaussian distributions will be introduced.

Keeping in mind that $|w\rangle$ resulted from the action of $(C(0.7) \otimes C(0.7) \otimes C(0.7))$ onto the $|000\rangle$ state an important pattern is observed when calculating the Hamming distances between $|000\rangle$ and all eight possible three-qubit states. Firstly, the state $|000\rangle$ has a Hamming distance of zero relative to itself and has the highest amplitude of 0.58566. Secondly, $|100\rangle$, $|010\rangle$ and $|001\rangle$ all have amplitudes of 0.11434 and a Hamming distance of one with respect to $|000\rangle$. Thirdly, $|110\rangle$, $|011\rangle$ and $|101\rangle$ have amplitudes of 0.02232 and a Hamming distance of two compared to $|000\rangle$. Lastly, $|111\rangle$ has a Hamming distance of three with respect to $|000\rangle$ and an amplitude of 0.00436. Hence, the amplitudes are always equal for equal Hamming distances and decrease with increasing Hamming distance.

To visualise the Gaussian distribution represented by $|w\rangle$, one simply needs to plot amplitudes against Hamming distances and mirror the resulting plot with respect to the vertical line connecting the horizontal zero value with the data point representing the highest amplitude. The corresponding plot is given in Fig. 5.9 and clearly shows a discrete Gaussian distribution centered around the binary qubit pattern with Hamming distance zero: in this case $|000\rangle$. It follows that applying the coin gate $C(0.7)$ to each qubit in an n-qubit systems $|q_1, q_2, ..., q_n\rangle$ yields a similar discretized Gaussian distribution centered around the binary qubit pattern $|q_1, q_2, ..., q_n\rangle$.
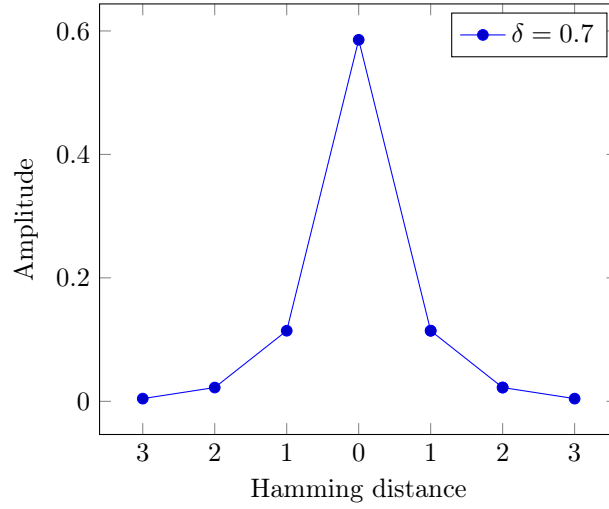
Figure 5.9: Plot visualizing the discrete Gaussian distribution of $|w\rangle$ resulting from acting coin gate C with $\delta = 0.7$ on each qubit in state $|000\rangle$. Amplitudes of the quantum state $|w\rangle$ (Equ. 5.63) are plotted against the Hamming distances between binary pattern $|000\rangle$ and all eight binary three-qubit pattern. The plot was mirrored with respect to the vertical line connecting the horizontal zero value with the data point representing the highest amplitude.

The observed pattern within the amplitudes can be generalized using Equ. 5.65: Acting the coin gate $C(\delta)$ on each qubit in the three-qubit state $|q_1, q_2, q_3\rangle$ will result in a discretized Gaussian distribution centered around the binary qubit pattern $|q_1, q_2, q_3\rangle$. The resulting amplitudes are dependent on the Hamming distance with respect to state $|q_1, q_2, q_3\rangle$: the state $|q_1, q_2, q_3\rangle$ with Hamming distance zero will have the largest amplitude of $\delta\sqrt{\delta}$, all states with Hamming distance one have amplitudes equal to $\delta - \delta\sqrt{\delta}$, states with Hamming distance two have amplitudes equal to $\sqrt{\delta}(\delta+1) - 2\delta$ and the state with the largest Hamming distance of three has the smallest amplitude of $-\sqrt{\delta}(3+\delta) + 1 + 3\delta$. A similar pattern will be observed when acting $C(\delta)$ on each qubit in any n-qubit state.

To demonstrate how different values of the parameter δ affect the shape of the resulting distribution, the coin gate $C(\delta)$ was applied to the four-qubit state $|0000\rangle$ using five different values for δ: 0.5, 0.6, 0.7, 0.8, 0.9. The five different distributions are shown in Fig. 5.10. The example uses the four-qubit state $|0000\rangle$ since it leads to a larger range on the horizontal axis thereby showing clearer differences between the distributions. Fig. 5.10 shows that the distributions flatten out with decreasing δ values. In the case of $\delta = 0.5$, the distribution has no visible peaks implying an equal superposition over all four-qubit states due to equal amplitudes.
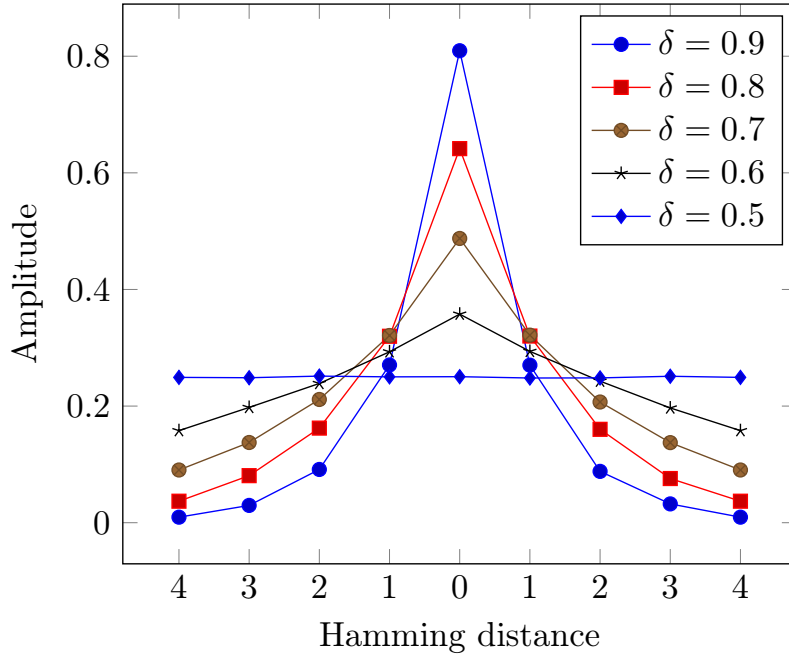
Figure 5.10: Plot visualizing the resulting discrete Gaussian distribution from the action of the coin gate $C(\delta)$ on each qubit in state $|0000\rangle$ for five different δ values. Amplitudes are plotted against the Hamming distances between binary pattern $|0000\rangle$ and all 16 binary four-qubit pattern. The plot was mirrored with respect to the vertical line connecting the horizontal zero value with the data point representing the highest amplitude.

To better illustrate the later classification problem another way of visualizing the Gaussian distribution needs to be introduced. This second tool for visualizing Hamming distances between three-qubit patterns is a three-dimensional cube as shown in Fig. 5.11. On that cube, adjacent qubit patterns have a Hamming distance (HD) of one and the Hamming distance increases by one with every additional corner. For example, the qubit state $|100\rangle$ is adjacent to $|110\rangle$ since they only differ in one qubit ($HD = 1$). Moving one more corner yields the state $|111\rangle$ or $|010\rangle$ which both have a Hamming distance of two when compared to $|100\rangle$.
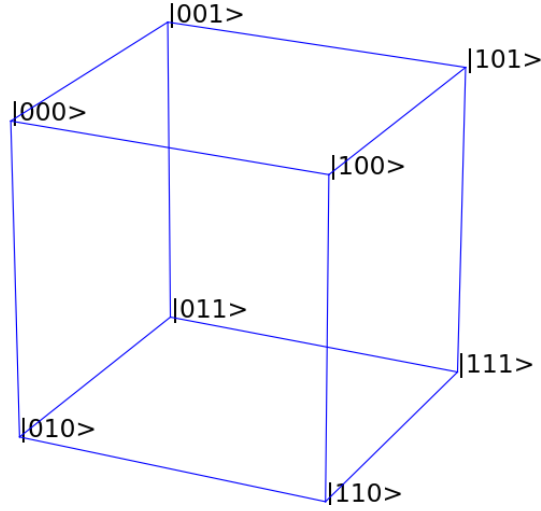
Figure 5.11: Visualizing Hamming distances on a three-dimensional cube. The Hamming distance between two binary qubit patterns is given by the shortest path on the edges of the cube between them. Thereby, the Hamming distance increases by one for every corner.

The cube can now be used as another way of visualizing the action of the coin gate $C(0.7)$ on each qubit in the $|000\rangle$ state. Before applying the C gates the qubit state is $|000\rangle$ with an amplitude of 1.0 shown in Fig. 5.12 wherein amplitude is denoted by a.

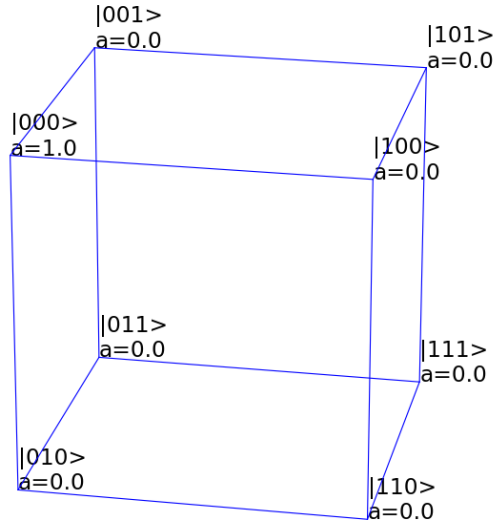Figure 5.12: The qubit state $|000\rangle$ with amplitude (a) of 1.0 visualized on a three-dimensional cube representing the Hamming distances between binary qubit patterns.

Applying the coin gate C to the first qubit redistributes the amplitudes between the $|000\rangle$ and $|100\rangle$ state which is visualized in Fig. 5.13 below.
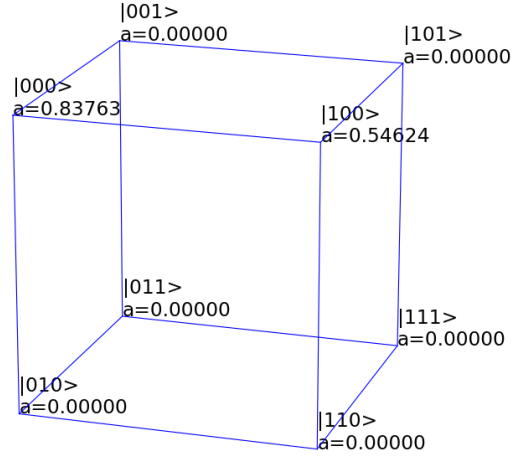
Figure 5.13: Visualization of the amplitude distribution over all three-qubit states after application of coin gate C to the first qubit of the state $|000\rangle$.

Fig. 5.14 illustrates the amplitude distribution after application of C gates to the first and second qubit in the state $|000\rangle$. It can be seen that at this point already four out of eight three-qubit states have non-zero amplitudes. With the application of the third C gate to the third qubit in $|000\rangle$ all eight-qubit patterns have non-zero amplitudes and the quantum state $|w\rangle$ from Equ. 5.63 is obtained. Fig. 5.15 shows this final quantum superposition representing a discrete Gaussian distribution over a three-dimensional cube based on Hamming distances. In this example, Fig. 5.13, 5.14 and 5.15 visually demonstrated a stepwise diffusion of the initial amplitude 1.0 of $|000\rangle$ over all eight three-qubit patterns on the cube.
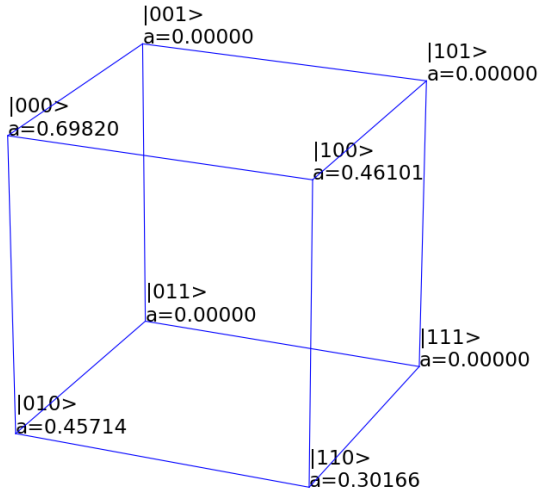
Figure 5.14: Visualization of the amplitude distribution over all three-qubit states after application of coin gate C to the first two qubits in the state $|000\rangle$.
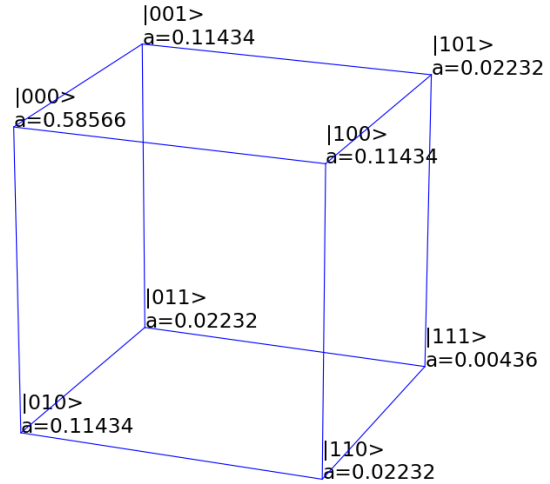
Figure 5.15: Final amplitude distribution over all three-qubit states after application of coin gate C to each qubit in the state $|000\rangle$ visualized on a three-dimensional cube.

With the familiarization with the coin gate C and the use of cubes to visualize amplitude distributions, the introduction of the tools needed to simulate the classification of Gaussian distributions with the amplitude-based kNN algorithm is completed. Yet, since any distance-weighted kNN algorithm is based on measuring distances between the input sample and each training sample, a new

distance metric for distributions needs to be introduced. For this reason, the Hellinger distance is defined in the red box below.

Definition: Hellinger distance

The Hellinger distance is a distance metric used to quantify the difference between any two probability distributions. In the case of continuous probability distributions it is defined using the Hellinger integral first proposed by Hellinger (1909). However, since the 2^n amplitudes of an n-qubit system will always constitute a discrete probability distribution only the Hellinger distance for discrete distributions will be introduced.

According to Prahladh (2011), the Hellinger distance between two discrete probability distributions $R = (r_1, ..., r_N)$ and $L = (l_1, ..., l_N)$, each with N entries, is defined as:

$$H(R, L) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{N} (\sqrt{r_i} - \sqrt{l_i})^2} \qquad (5.67)$$

and can be rewritten in terms of the Euclidean distance between the square root vectors $\sqrt{R}$ and $\sqrt{L}$:

$$H(R, L) = \frac{1}{\sqrt{2}} \| \sqrt{R} - \sqrt{L} \|_2 \qquad (5.68)$$

Furthermore, the squared Hellinger distance can be expressed as

$$H^2(R, T) = \frac{1}{\sqrt{2}} \sum_{i=1}^{N} (\sqrt{r_i} - \sqrt{t_i})^2 \qquad (5.69)$$

Consequently, the amplitude-based kNN algorithm should ideally classify Gaussian distributions based on their Hellinger distance and not based on Euclidean distance. Yet, Eq. 5.67 and Eq. 5.68 revealed a strong relationship between the two distance metrics. This fact can be exploited for the use in the amplitude-based kNN algorithm.

As previously discussed in Section 5.2, the probability of measuring a certain class, e.g. $|1\rangle$, after the execution of the amplitude-based kNN algorithm is given by

$$Prob(|c^m\rangle = |1\rangle) = \sum_{m|c^m=1} 1 - \frac{1}{4M} \sum_{i=1}^{N} | x_i - t_i^m |^2 \qquad (5.70)$$

Suppose the probability distribution $R = (r_1, ..., r_N)$ is the input sample that shall be classified and the distributions $L^m = (l_1^m, ..., l_N^m)$ are the training samples. Instead of directly using them for the amplitude-based kNN algorithm, one chooses their square root vectors such that $x_i \to \sqrt{r_i}$ and $t_i^m \to \sqrt{l_i^m}$. Eq. 5.70 then reads

$$Prob(|c^m\rangle = |1\rangle) = \sum_{m|c^m=1} 1 - \frac{1}{4M} \sum_{i=1}^{N} | \sqrt{r_i} - \sqrt{l_i^m} |^2 \qquad (5.71)$$

which is now clearly dependent on the squared Hellinger distance between the input distribution R and each training distribution L^m.

It remains to discuss how to load the input and training distributions into the initial quantum state (Eq. 5.14) required for the amplitude-based kNN algorithm. The neccessary procedure is outlined in Appendix A.

At this point, all tools needed to classify Gaussian distributions with the amplitude-based kNN algorithm have been discussed.

For the beginning, the classification of Gaussian distributions with eight entries requiring three qubits will be

ID	Gaussian distr. centered around	$Prob(CM)$	$Prob$ $(\lvert c\rangle = \lvert 0\rangle)$	$Prob$ $(\lvert c\rangle = \lvert 1\rangle)$	Expected class	Algorithm output
1	$\lvert 100\rangle$	$0.9930°$	$0.4904°$	$0.5096°$	$\lvert 0\rangle$	$\lvert 1\rangle$ °
		$0.8090^{\bullet}$	$0.5414^{\bullet}$	$0.4586^{\bullet}$		$\lvert 0\rangle$ $^{\bullet}$
		$0.6740^{\triangleright}$	$0.5549^{\triangleright}$	$0.4451^{\triangleright}$		$\lvert 0\rangle$ $^{\triangleright}$
2	$\lvert 010\rangle$	$0.9830°$	$0.4914°$	$0.5086°$	$\lvert 0\rangle$	$\lvert 1\rangle$ °
		$0.8050^{\bullet}$	$0.5205^{\bullet}$	$0.4795^{\bullet}$		$\lvert 0\rangle$ $^{\bullet}$
		$0.6600^{\triangleright}$	$0.5636^{\triangleright}$	$0.4364^{\triangleright}$		$\lvert 0\rangle$ $^{\triangleright}$
3	$\lvert 001\rangle$	$0.8080°$	$0.4980°$	$0.5020°$	$\lvert 0\rangle$	$\lvert 1\rangle$ °
		$0.8120^{\bullet}$	$0.5582^{\bullet}$	$0.4418^{\bullet}$		$\lvert 0\rangle$ $^{\bullet}$
		$0.6700^{\triangleright}$	$0.5642^{\triangleright}$	$0.4358^{\triangleright}$		$\lvert 0\rangle$ $^{\triangleright}$
4	$\lvert 110\rangle$	$0.9830°$	$0.4914°$	$0.5086°$	$\lvert 1\rangle$	$\lvert 1\rangle$ °
		$0.8040^{\bullet}$	$0.4726^{\bullet}$	$0.5274^{\bullet}$		$\lvert 1\rangle$ $^{\bullet}$
		$0.6350^{\triangleright}$	$0.4520^{\triangleright}$	$0.5480^{\triangleright}$		$\lvert 1\rangle$ $^{\triangleright}$
5	$\lvert 011\rangle$	$0.9830°$	$0.4964°$	$0.5036°$	$\lvert 1\rangle$	$\lvert 1\rangle$ °
		$0.8300^{\bullet}$	$0.4398^{\bullet}$	$0.5602^{\bullet}$		$\lvert 1\rangle$ $^{\bullet}$
		$0.6730^{\triangleright}$	$0.4383^{\triangleright}$	$0.5617^{\triangleright}$		$\lvert 1\rangle$ $^{\triangleright}$
6	$\lvert 101\rangle$	$0.9840°$	$0.4888°$	$0.5112°$	$\lvert 1\rangle$	$\lvert 1\rangle$ °
		$0.8110^{\bullet}$	$0.4451^{\bullet}$	$0.5549^{\bullet}$		$\lvert 1\rangle$ $^{\bullet}$
		$0.6320^{\triangleright}$	$0.4320^{\triangleright}$	$0.5680^{\triangleright}$		$\lvert 1\rangle$ $^{\triangleright}$

Table 5.12: Amplitude-based kNN algorithm classification results for various Gaussian distributions centered around six different three-qubit patterns. The Gaussian distributions were created using the coin gate $C(\delta)$. For each pattern three different distributions were prepared and classified by changing the parameter δ: The results for $\delta = 0.6$ are always on top and marked with an empty circle (°), results for $\delta = 0.85$ are always in the middle marked with a filled circle (•) and results for $\delta = 0.95$ are always at the bottom and marked with a triangle (▷)

Chapter 6

Outlook

Chapter 7

Conclusion

Chapter 8

Personal Reflection

Personally, this bachelor thesis research has been instructive and beneficial in many different ways. Most importantly, I have had the opportunity to dedicate my entire time to learn about the subject of quantum information and, specifically, quantum machine learning in detail. Since these subjects are not taught within the curriculum of the Maastricht Science Programme (MSP) it was especially amazing to challenge myself to these notoriously difficult subjects within the intersection of quantum physics and computer science. In doing so, I have learned more about the methods of theoretical physics and gained additional experience in scientific programming with Octave, Python and F#. Without prior knowledge of F#, I was able to learn how to simulate quantum computations and quantum machine learning algorithms using the quantum simulation toolsuite Liqui|⟩. Furthermore, I was able to use the first cloud-based quantum computer, the so-called IBM Quantum Experience, publicly released in May 2016 by IBM.

Alongside my research, I had the possibility to attend many great lectures, seminars and three conferences which were all generously funded by the Centre for Quantum Technology. This enabled me to get to know many renowned scientists working in the fields of quantum information, quantum machine learning, open quantum systems, quantum optics and quantum cryptography. Furthermore, I was provided with the opportunity to present my research at the 4th South African conference for quantum information processing, communication and control in Cape Town in late November 2016. This constituted a big step with respect to my academic career and a great way of putting my acquired presentation skills to practice.

However, in retrospective I feel that I could have been more productive in office by structuring my work days more diligently. For example, the first one and a half months were mostly spent on reading research paper and text books to acquire the theoretical foundation for my research and little time was used to practice F# within the Liqui|⟩ framework. A possible solution for the future would be to divide each day into two parts: e.g. spending the morning with reading papers and books and the afternoon on practical work and programming. Besides this, I feel like I should have communicated my progress more with my internal supervisor Dr. Birembaut at the MSP.

In conclusion, the bachelor thesis research has showed me the importance of interdisciplinarity since the field of quantum information requires the understanding of concepts in computer science as well as quantum mechanics. Fortunately, MSP's liberal education does not only focus on textbook exercises and lectures but mostly teaches us how to systematically approach unknown topics and tackle problems therein. It was great to see that this enabled me to venture into a previously unfamiliar field and learn the necessary skills to conduct meaningful research. The newly acquired skills will certainly be useful for my planned masters in the field of quantum information.

Appendix A

Quantum state preparation with Gaussian distributions

To keep the discussion simple, the task is to load one Gaussian input distribution R and two Gaussian training samples, L_1 and L_2, such that the amplitude-based kNN algorithm can be used to classify R. Note that the input and training distributions are restricted to Gaussian distributions that can be initialized by using the coin gate $C(\delta)$ for some value of δ. The input and training samples all have N entries whereby N is restricted to be a multiple of two. To encode the distributions, n data qubits are required such that there are $2^n = N$ amplitudes. Additionally, one ancilla, one class and one m qubit are needed. Since all qubits are initialised to the $|0\rangle$ state, the initial state is

$$|\Upsilon_0\rangle = |a; d_1, ..., d_n; c; m\rangle = |0; 0_1, ..., 0_n; 0; 0\rangle \tag{A.1}$$

where the first register holds the ancilla (a), the second register holds the n data qubits (d) used to encode the distributions and the third and fourth register consists of the class (c) and m-qubit respectively.

As already demonstrated in Eq. 5.26 and Eq. 5.27 in Section 5.2.1, the ancilla and m-register are each put into an equal superposition through the application of two H gates. The state is then

$$|\Upsilon_1\rangle = \frac{1}{2} \sum_{m=0}^{1} \big[|0\rangle |0_1, ..., 0_n\rangle + |1\rangle |0_1, ..., 0_n\rangle \big] |0\rangle |m\rangle \tag{A.2}$$

Suppose the Gaussian distribution R is obtained by choosing $\delta = \tau$ for the coin gate $C(\delta)$. Then the next step is to make use of the controlled version of the coin gate $CC(\tau)$ controlled by the ancilla qubit. $CC(\tau)$ is applied to each of the n qubits in the data register to load the input distribution R. Hence, the new quantum state is

$$|\Upsilon_2\rangle = \prod_{j=1}^{n} CC(\tau)(a, d_j) |\Upsilon_1\rangle = \frac{1}{2} \sum_{m=0}^{1} \big[|0\rangle |0_1, ..., 0_n\rangle + |1\rangle |R\rangle \big] |0\rangle |m\rangle \tag{A.3}$$

Since the algorithm is simulated in Liqui$|\rangle$, it is straightforward to define the controlled version of the coin gate $C(\delta)$.

Next, using an X gate to flip the ancilla moves the input distribution R onto the $|0\rangle$ state of the ancilla such that the state is now

$$|\Upsilon_3\rangle = X(a)|\Upsilon_2\rangle = \frac{1}{2}\sum_{m=0}^{1}\big[\,|0\rangle|R\rangle + |1\rangle|0_1,...,0_n\rangle\,\big]|0\rangle|m\rangle \tag{A.4}$$

Suppose the first training distribution L_1 can be loaded by choosing $\delta = \sigma$ and applying $C(\sigma)$ to the qubit pattern $|0_1,...,1_5,0_6,...,0_n\rangle$. Thus, the fifth qubit in the data register connected to the $|1\rangle$ ancilla state needs to flipped into the $|1\rangle$ state. This can be done with a CCNOT gate, controlled by the ancilla and m qubit, acting on the fifth data qubit. Next, by applying the controlled controlled version of the coin gate $CCC(\sigma)$ to each data qubit controlled by the ancilla and the m qubit the training sample L_1 is loaded. The quantum state is then given by

$$\begin{aligned}|\Upsilon_4\rangle &= \prod_{j=1}^{n} CCC(\sigma)(a,m,d_j)CCNOT(a,m,d_5)|\Upsilon_3\rangle\\ &= \frac{1}{2}\Big[\big[\,|0\rangle|R\rangle + |1\rangle|0_1,...,0_n\rangle\,\big]|0\rangle|0\rangle + \big[\,|0\rangle|R\rangle + |1\rangle|L_1*\rangle\,\big]|0\rangle|1\rangle\,\Big]\end{aligned} \tag{A.5}$$

Since the last entry on the diagonal of the coin gate (see Eq. 5.64) is negative, the application of $CCC(\sigma)$ introduces a negative sign when applied to the fifth data qubit. Hence, not the exact distribution L_1 but a distribution L_1^* with a sign difference was loaded into $|\Upsilon_4\rangle$. As explained in Section 5.2, the amplitude-based kNN algorithm makes use of interference between the input and training samples whereby the negative sign would alter the outcome of the interference. Thus, the sign needs to be corrected by acting a controlled controlled Z gate on the fifth data qubit. Thereafter, the m qubit is flipped with an X gate to move L_1 onto the $|0\rangle$ state of the m qubit. After these gate operations the state is

$$\begin{aligned}|\Upsilon_5\rangle &= X(m)CCZ(d_5)|\Upsilon_4\rangle\\ &= \frac{1}{2}\Big[\big[\,|0\rangle|R\rangle + |1\rangle|L_1\rangle\,\big]|0\rangle|0\rangle + \big[\,|0\rangle|R\rangle + |1\rangle|0_1,...,0_n\rangle\,\big]|0\rangle|1\rangle\,\Big]\end{aligned} \tag{A.6}$$

For the second training distribution L_2, suppose it results from choosing $\delta = \nu$ and applying $C(\nu)$ to the qubit pattern $|1_1,...,1_n\rangle$. Thus, all data qubits connected to the $|1\rangle$ ancilla and $|1\rangle$ m-qubit state have to be flipped using a CCNOT gates. As outlined in the previous step, $CCC(\nu)$ loads the distribution L_2^* and not L_2 because of sign differences. Since all data qubits are $|1\rangle$, these signs are corrected by acting controlled controlled Z gates on all data qubits controlled by the ancilla and m-qubit. The state containing the input and both training distributions is then

$$\begin{aligned}|\Upsilon_6\rangle &= \prod_{j=1}^{n} CCZ(d_j)CCC(\sigma)(a,m,d_j)CCNOT(a,m,d_j)|\Upsilon_5\rangle\\ &= \frac{1}{2}\Big[\big[\,|0\rangle|R\rangle + |1\rangle|L_1\rangle\,\big]|0\rangle|0\rangle + \big[\,|0\rangle|R\rangle + |1\rangle|L_2\rangle\,\big]|0\rangle|1\rangle\,\Big]\end{aligned} \tag{A.7}$$

Lastly, the class qubit for L_2 is flipped with a CNOT gate controlled by the m qubit such that the final state is

$$\begin{aligned}|\Upsilon_6\rangle &= CNOT(m,c)|\Upsilon_6\rangle\\ &= \frac{1}{2}\Big[\big[\,|0\rangle|R\rangle + |1\rangle|L_1\rangle\,\big]|0\rangle|0\rangle + \big[\,|0\rangle|R\rangle + |1\rangle|L_2\rangle\,\big]|1\rangle|1\rangle\,\Big]\end{aligned} \tag{A.8}$$

$|\Upsilon_7\rangle$ is in the form of the initial quantum state (Eq. 5.14) required for the amplitude-based kNN algorithm. This completes the quantum state preparation routine for loading a restricted subset of discrete Gaussian distributions using the coin gate $C(\delta)$.

References

Bachmann, P. (1894). *Die analytische Zahlentheorie* (Vol. 2). Teubner.

Bekkerman, R., Bilenko, M., & Langford, J. (2011). *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press.

Bishop, L. (2016). *Quantum gate times on IBM quantum computer*. Retrieved 2016-12-29, from `https://quantumexperience.ng.bluemix.net/qstage/#/community/question?questionId=71b344418d724f8ec1088bafc75eb334&answerId=3a2f15c989b2aa752f563cfaf53ae240`

Booth Jr, J. (2012). Quantum compiler optimizations. *arXiv preprint arXiv:1206.3348*.

Cai, X. D., Wu, D., Su, Z. E., Chen, M. C., Wang, X. L., Li, L., ... Pan, J. W. (2015). Entanglement-based machine learning on a quantum computer. *Physical Review Letters*, *114*(11), 1–5. doi: 10.1103/PhysRevLett.114.110504

Dawson, C. M., & Nielsen, M. A. (2005). The Solovay-Kitaev algorithm. *arXiv preprint quant-ph/0505030*.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78–87.

D-Wave. (2015). *Breaking through 1000 Qubits*. `http://www.dwavesys.com/blog/2015/06/breaking-through-1000-qubits`. (Accessed: 2016-09-09)

Giovannetti, V., Lloyd, S., & Maccone, L. (2008). Quantum random access memory. *Physical review letters*, *100*(16), 160501.

Grover, L. K., & Rudolph, T. (2002). Creating superpositions that correspond to efficiently integrable probability distributions. *Arxiv preprint*, 2. Retrieved from `http://arxiv.org/abs/quant-ph/0208112`

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System technical journal*, *29*(2), 147–160.

Hellinger, E. (1909). Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen. *Journal für die reine und angewandte Mathematik*, *136*, 210–271.

IBM. (2016a). *The IBM Quantum Experience*. Retrieved 2016-12-29, from `http://www.research.ibm.com/quantum/`

IBM. (2016b). *What is big data?* Retrieved 2016-09-08, from `https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html`

Kitaev, A. Y. (1995). Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*.

Las Heras, U., Alvarez-Rodriguez, U., Solano, E., & Sanz, M. (2016). Genetic algorithms for digital quantum simulations. *Physical Review Letters*, *116*(23), 230504.

Li, Z., Liu, X., Xu, N., & Du, J. (2015). Experimental realization of a quantum support vector machine. *Physical Review Letters*, *114*(14), 1–5. doi: 10.1103/PhysRevLett.114.140504

Microsoft Research. (2016). *Language-Integrated Quantum Operations: LIQUi|>*. Retrieved 2016-12-29, from `https://www.microsoft.com/en-us/research/project/language-integrated-quantum-operations-liqui/`

Nielsen, M. A., & Chuang, I. L. (2010). *Quantum computation and quantum information*. Cambridge university press.

O'Malley, P. J. J., Babbush, R., Kivlichan, I. D., Romero, J., McClean, J. R., Barends, R., . . . Martinis, J. M. (2016, Jul). Scalable Quantum Simulation of Molecular Energies. *Phys. Rev. X*, *6*, 031007. Retrieved from `http://link.aps.org/doi/10.1103/PhysRevX.6.031007` doi: 10.1103/PhysRevX.6.031007

Prahladh, H. (2011). *Lecture notes on communication complexity.* Retrieved 2017-01-06, from `http://www.tcs.tifr.res.in/~prahladh/teaching/2011-12/comm/lectures/l12.pdf`

Ristè, D., da Silva, M. P., Ryan, C. A., Cross, A. W., Smolin, J. A., Gambetta, J. M., . . . Johnson, B. R. (2015). Demonstration of quantum advantage in machine learning. *arXiv:1512.06069*, 1–12. Retrieved from `http://arxiv.org/abs/1512.06069`

Schuld, M., Fingerhuth, M., & Petruccione, F. (2016). Amplitude-based quantum k-nearest neighbour algorithm. Manuscript in preparation.

Schuld, M., Sinayskiy, I., & Petruccione, F. (2014). Quantum computing for pattern classification. , 14. Retrieved from `http://arxiv.org/abs/1412.3646` doi: 10.1007/978-3-319-13560-1

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of computer science, 1994 proceedings., 35th annual symposium on* (pp. 124–134).

Soklakov, A. N., & Schack, R. (2006). Efficient state preparation for a register of quantum bits. *Physical Review A*, *73*(1), 012307.

Trugenberger, C. (2001). Probabilistic Quantum Memories. *Physical Review Letters*, *87*(6), 067901. Retrieved from `http://arxiv.org/abs/quant-ph/0012100` doi: 10.1103/PhysRevLett.87.067901

Wecker, D., & Svore, K. M. (2014). *LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing.* Retrieved from `arXiv:1402.4467v1`