

Mark Fingerhuth

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

Blah blah ...

Contents

Abstract	v
Nomenclature	vii
1 Introduction	1
1.1 Motivation	2
1.2 Research Question	2
2 Theoretical Foundations	3
2.1 Quantum Bits	3
2.1.1 Single Qubit Systems	3
2.1.2 Multiple Qubit Systems	6
2.2 Quantum Logic Gates	8
2.2.1 Single Qubit Gates	8
2.2.2 Multiple Qubit Gates	11
2.3 Classical Machine Learning	12
2.3.1 k-nearest Neighbour Algorithm	12
2.3.2 Algorithmic Time Complexity	14
3 Methods	17
3.1 Liqui ⟩	17
3.2 IBM Quantum Experience	18
4 Literature Review: Quantum-enhanced Machine Learning	20
4.0.1 Quantum State Preparation	20
4.0.2 Quantum k-nearest Neighbour Algorithm	24
5 Results and Discussion	28
5.1 Simulating the qubit-based kNN algorithm	28
5.2 Development of an amplitude-based kNN algorithm	33
5.2.1 Implementing the amplitude-based kNN algorithm	34
5.2.2 Simulating the amplitude-based kNN algorithm	45
6 Outlook	47
7 Conclusion	48
8 Personal reflection	49
References	50

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. $3! = 3*2*1$

Indices

a	Ambient
air	Air

Acronyms and Abbreviations

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

Chapter 1

Introduction

SOME MORE GENERAL INTRO TEXT HERE?

The ability to understand spoken language, to recognize 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 quintillion (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 special properties to manipulate and process information in ways that are impossible to implement on classical computers. The quantum equivalent to 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 sort of 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 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

Don't know where this should go!

Matthias Troyer citation for this part:

There are many different ways of realising a QC such as using trapped ions, single photon sources or superconducting Josephson junctions, etc. Depending on the chosen substrate, different sets of quantum logic gates can be implemented and in order to run a quantum algorithm it has to be mapped to the available hardware. Thus, quantum algorithms have to be translated (compiled) into a series of gates consisting only of quantum gates from the available gate set. This is referred to as *quantum compilation*.

Finding optimal ways of compiling a given quantum algorithm as well as designing a high-level programming language or environment similar to that of classical computer code editors is called *quantum software engineering*.

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

How can a theoretically proposed 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 in order to answer this research question.

Chapter 2

Theoretical Foundations

CHECK FOOTNOTE NUMBERS IN WHOLE DOCUMENT! INTRODUCE THE NOTION OF QUANTUM REGISTER SOMEWHERE!

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 for 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. 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.

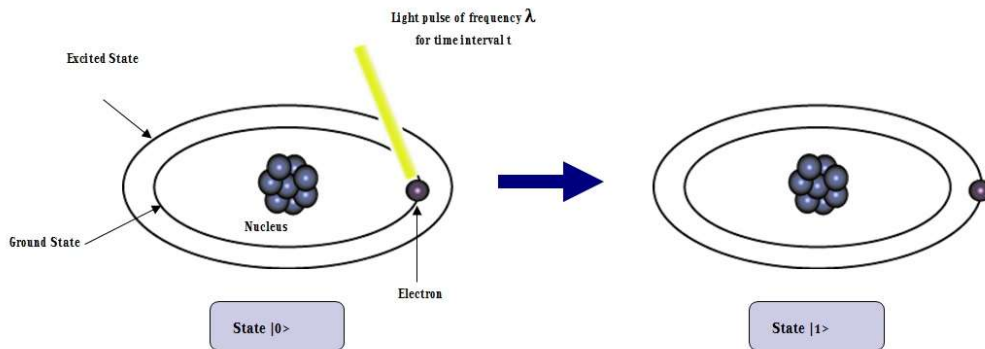


Figure 2.1: A simple example of a physical qubit using the electron of a hydrogen atom

¹Reprinted from RF Wireless World, n.d., Retrieved December 23, 2016, from <http://www.rfwireless-world.com/Terminology/Difference-between-Bit-and-Qubit.html>. Copyright 2012 by RF Wireless World. Reprinted with permission.

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 powerful 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:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \quad (2.1)$$

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,

$$\eta = e^{i\phi} e \quad (2.2)$$

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!

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (2.3)$$

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

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

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

$$Prob(|0\rangle) = |\alpha|^2 \quad (2.5)$$

and $|1\rangle$ with a probability of

$$Prob(|1\rangle) = |\beta|^2 \quad (2.6)$$

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:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2.7)$$

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:

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

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.

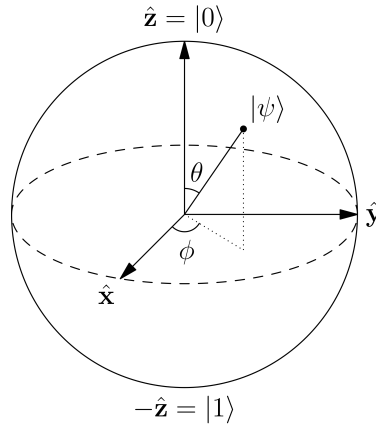


Figure 2.2: An arbitrary two-dimensional qubit $|\psi\rangle$ visualized on the Bloch sphere.²

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:

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

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 normalization of the qubit state it is acting on.

²Reprinted from Wikipedia, n.d., Retrieved September 7, 2016, from https://en.wikipedia.org/wiki/Bloch_Sphere. Copyright 2012 by Glosser.ca. Reprinted with permission.

$$a) | \det(U) | = 1 \quad (2.10)$$

$$b) UU^\dagger = U^\dagger U = \mathbb{1} = UU^{-1} = U^{-1}U \quad (2.11)$$

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, \dots, 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 Multiple Qubit Systems

A classical computer with one bit of memory is not particularly useful and equally a QC with one qubit is rather useless. In order 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:

$$|\psi\rangle \otimes |\psi\rangle = |0\rangle \otimes |0\rangle = |00\rangle \quad (2.12)$$

whereby the last expression omits the \otimes symbol which is the shorthand form of a tensor product between two qubits.

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

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

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 for 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,

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

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:

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

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

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

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.

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

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

This can be factorized into the following tensor product:

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

In contrast, consider the following state:

$$|\phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad (2.19)$$

It is straightforward to see that there is no way to factorize 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:

$$|\phi\rangle_{PM} = |11\rangle \quad (2.20)$$

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 suprising 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 visualized 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:

$$\sigma_1 = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (2.21)$$

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:

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

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,

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

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,

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

On the Bloch sphere, the X gate corresponds to an anti-clockwise π rotation around the x-axis as shown in Fig. 2.3.

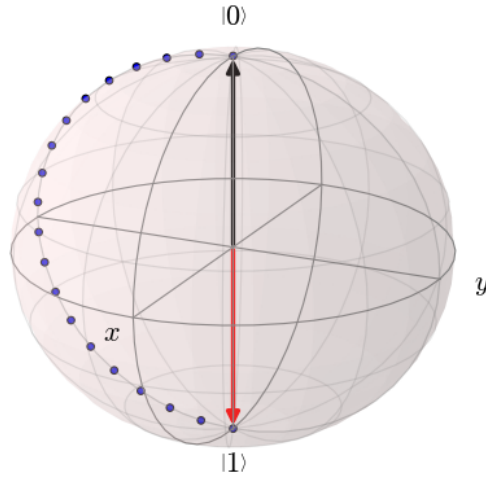


Figure 2.3: Visualization of the X gate on the Bloch sphere. Initial state (black) transformed to final state (red).

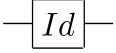
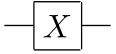
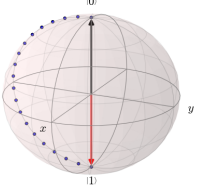
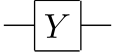
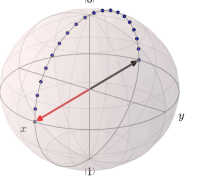
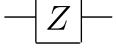
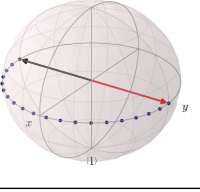
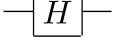
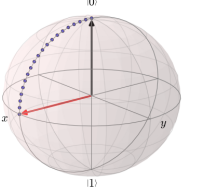
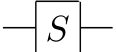
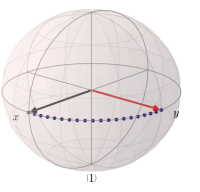


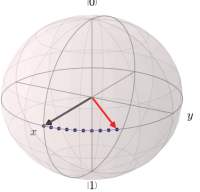


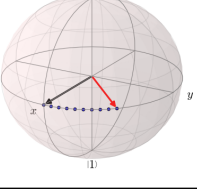

From Equ. 2.22, 2.23 and 2.24 it follows that X is its own inverse as well as its own Hermitian conjugate:

$$XX = XX^\dagger = \mathbb{1} \quad (2.25)$$

$$X = X^{-1} = X^\dagger \quad (2.26)$$

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

2.2.2 Multiple 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:

$$CNOT = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.27)$$

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 parantheses 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,

$$|\phi\rangle = |0\rangle \otimes |0\rangle = |00\rangle \quad (2.28)$$

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

$$(H \otimes \mathbb{1}) |\phi\rangle = (H \otimes \mathbb{1}) |00\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |10\rangle \quad (2.29)$$

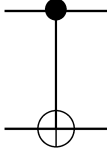
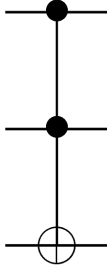
Now consider applying the CNOT to this state whereby the control qubit is coloured **red** and the target qubit **green**.

$$CNOT(\text{red}, \text{green}) \otimes \left(\frac{1}{\sqrt{2}} |\text{red}0\rangle + \frac{1}{\sqrt{2}} |\text{red}1\rangle \right) = \frac{1}{\sqrt{2}} |\text{red}0\rangle + \frac{1}{\sqrt{2}} (\mathbb{1} \otimes X) |\text{red}1\rangle = \frac{1}{\sqrt{2}} |\text{red}0\rangle + \frac{1}{\sqrt{2}} |\text{red}1\rangle \quad (2.30)$$

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^{th} control qubit and $\mathbb{1}_k$ for the $k \times k$ identity matrix.

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

2.3 Classical Machine Learning

Machine learning can be subdivided into three major fields.

Supervised ML

- Based on *input* and *output* data

"I know how to classify this data but I need the 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 is focussing on quantum-enhancements in the field of supervised machine learning only.

2.3.1 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 5 different pictures of apples and 5 different pictures of bananas. This will be called the *training data set* D_T . The pictures in D_T may be taken from different angles, in varying light settings and include different coloured apples and bananas. Two examples of such pictures 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 j^{th} picture are stored in the m -dimensional *feature vector* \vec{v}_j . Mathematically, the training data set D_T consists of ten feature vectors $\vec{v}_0, \vec{v}_1, \dots, \vec{v}_{10}$ that are each assigned to either class A (apple) or B (banana). The training vectors are visualized 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).

³Reprinted 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.

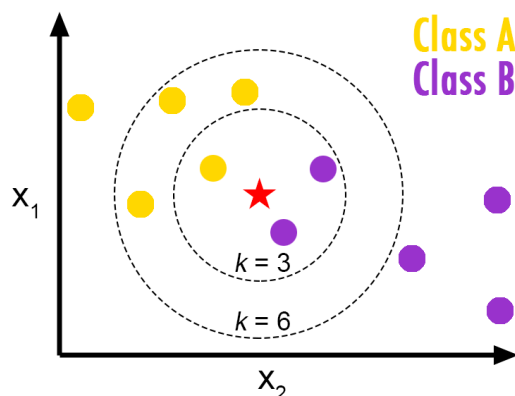


Figure 2.5: Visualization of a kNN classifier⁴

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.

⁴Reprinted 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.

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

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

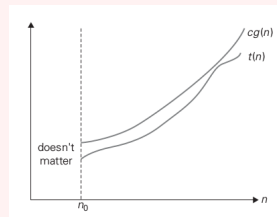


Figure 2.6: Visualization of $c * g(n)$ being an upper asymptotic bound of $t(n)$ ⁵

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 $n \rightarrow \infty$.

⁵Reprinted 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.

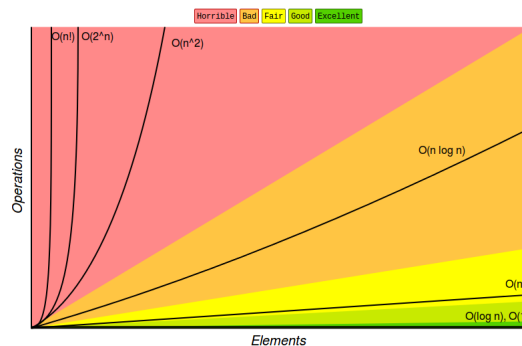


Figure 2.7: Overview of algorithmic complexity classes⁶

⁶Reprinted from Big-O Cheatsheet, Retrieved December 28, 2016, from <http://bigocheatsheet.com/>. Copyright

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 library⁵ for Python was used to create the Bloch sphere plots. All function plots were embedded directly into L^AT_EX by using the pgfplots package⁶.

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 Microsoft Research will be used. Liqui| \rangle is based on the functional programming language F# and allows for simulation of up to 30 qubits (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.

⁵The open-source QuTiP library for Python may be downloaded from <http://qutip.org/>

⁶The L^AT_EXpgfplots 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|>

A Lenovo ThinkPad T450 containing an Intel i5 processor with 2 cores and 8GB random-access memory (RAM) is used for all quantum simulations in this thesis.

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 provides the user with access to the "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 10 quantum logic gates.

DISCUSS THE AVAILABLE GATES HERE!

Additionally, there are two different measurement types: 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.

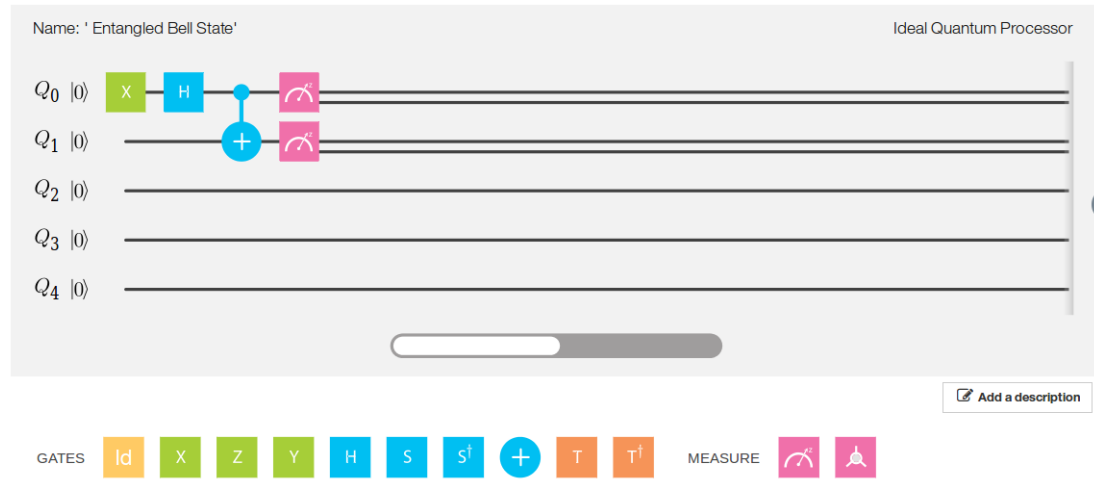


Figure 3.2: Screenshot showing IBM Quantum Composer

By spending limited user coins the gate sequence of a composed algorithm is then sent 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 $52.3\mu\text{s}$ to $81.5\mu\text{s}$. Furthermore, the phase damping times range from $60.9\mu\text{s}$ to $112.4\mu\text{s}$. Currently, the implementation of a single qubit quantum logic gate takes 130ns and applying a CNOT gate takes 500ns (Bishop, 2016).

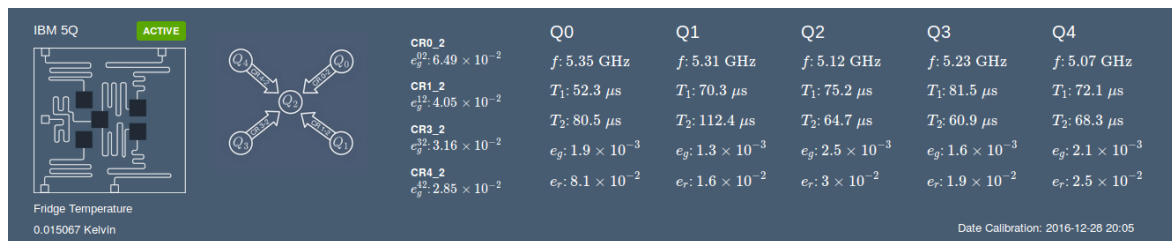


Figure 3.3: Screenshot of IBM QC calibration results (28. December 2016 - 20:05)

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 only be executed on a quantum computer.

C/Q is the topic of this thesis

Analogously, Q/C is the subfield where quantum data is being processed using a classical machine learning algorithm. Examples for Q/C:

used genetic algorithms to reduce digital and experimental errors in quantum gates

A tantamount task is then to find a model (a.k.a. effective) Hamiltonian of the system and to determine properties of the present noise sources. By computing likelihood functions in an adaptation of Bayesian inference, Wiebe et al. found that quantum Hamiltonian learning can be performed using realistic resources such as depolarizing noise.

Q/Q would include learning a model Hamiltonian of a system implemented in a lab using a quantum learning algorithm.

4.0.1 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 :

$$v = \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \quad (4.1)$$

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,

$$\begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} * 10 = \begin{pmatrix} 6 \\ 4 \end{pmatrix} \quad (4.2)$$

Convert each entry to binary,

$$\begin{pmatrix} 6 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} 0110 \\ 0100 \end{pmatrix} \quad (4.3)$$

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

$$\begin{pmatrix} 0110 \\ 0100 \end{pmatrix} \rightarrow n = 01100100 \quad (4.4)$$

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

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

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. Only slight (up to quadratic) speed-ups are possible through clever quantum algorithm design (CITATION).

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

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

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 ,

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

and introduces its controlled version CS^j :

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

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.

$$|\Psi_0^1\rangle = |l^1; u; m\rangle = |l_1^1, l_1^1, \dots, l_n^1; 01; 0_1, \dots, 0_n\rangle \quad (4.9)$$

The routine will use the second utility qubit u_2 to separate the initial 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 ($u_2 = |1\rangle$):

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

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:

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

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):

$$|\Psi_3^j\rangle = nCNOT(m_1, m_2, \dots, m_n, u_2) |\Psi_2^j\rangle \quad (4.12)$$

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 ($u_2 = |0\rangle$):

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

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:

$$|\Psi_5^j\rangle = nCNOT(m_1, m_2, \dots, m_n, u_2) |\Psi_4^j\rangle \quad (4.14)$$

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

The resulting state is now given by the following equation:

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

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

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

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.

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.

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

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.

4.0.2 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, \dots, v_n each assigned to a class c as follows,

$$|T\rangle = \frac{1}{\sqrt{N}} \sum_{p=1}^N |v_1^p, v_2^p, \dots, v_n^p, c^p\rangle \quad (4.19)$$

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

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

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

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

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,

$$\begin{aligned}
|\psi_2\rangle &= \frac{1}{\sqrt{N}} \sum_{p=1}^N |x_1, x_2, \dots, x_n; v_1^p, v_2^p, \dots, v_n^p; c^p\rangle \otimes H|0\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{p=1}^N |x_1, x_2, \dots, x_n; v_1^p, v_2^p, \dots, v_n^p; c^p\rangle \otimes \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}
\end{aligned} \tag{4.22}$$

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 (CITATION?)

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

Given quantum state $|\psi_2\rangle$ the HD between the input and each training register can be calculated by applying $CNOT(x_s, v_s^p)$ 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,

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

By applying the unitary operator U defined by,

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

where

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

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 p th training vector \vec{v}^p and the input vector \vec{x} is written into the complex phase of the p th 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,

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

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,

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

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:

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

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:

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

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

Some more general text Why are some parts simulating and some parts using IBM? All the code can be found on GitHub⁷

5.1 Simulating the qubit-based kNN algorithm

The computer used for the Liqui⁷ 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,

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

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.

⁷Link to GitHub folder.







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

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 red and blue 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.




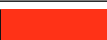






Colour	Binary 9-bit RGB string	Class
	111 000 000	red ($ 0\rangle$)
	101 000 000	red ($ 0\rangle$)
	110 000 000	red ($ 0\rangle$)
	111 100 100	red ($ 0\rangle$)
	111 000 100	red ($ 0\rangle$)
	000 000 111	blue ($ 1\rangle$)
	000 000 101	blue ($ 1\rangle$)
	000 000 100	blue ($ 1\rangle$)
	100 000 111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

Table 5.3: Training data set II

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 4.0.2 is preparing the initial superposition over all six 9-bit RGB training patterns t^j :

$$|T\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^N |t_1^j, t_2^j, \dots, t_9^j; c^j\rangle \quad (5.2)$$

This can be done using the quantum state preparation algorithm by Trugenberger (2001) outlined in Section 4.0.1. In this case, the seven steps (see blue box in Section 4.0.1) 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,





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









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$)
	110 100 110	red ($ 0\rangle$)
	000 000 110	blue ($ 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

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

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 $|\psi_0\rangle$:

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

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 $|\phi_1\rangle$:

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

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 8th qubit through the application of an X gate. The state is now given by,

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

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 $3\text{CNOT}(t_7, t_8, t_9, c)$ 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 $3\text{CNOT}(t_7, t_8, t_9, c)$ 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 $9\text{CNOT}(t_1, t_2, \dots, t_9, c)$ 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,

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

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.

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

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 $\text{CNOT}(x_s, t_s^j)$ gates. By applying an X gate to every qubit in the fourth register the HD gets reversed as discussed in Section 4.0.2. The state is now,

$$|\phi_5\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^6 \prod_{s=1}^9 X(t_s^j) \text{CNOT}(x_s, t_s^j) \left[|x_1, x_2, \dots, x_9; c^j; 0; t_1^j, t_2^j, \dots, t_9^j\rangle + |x_1, x_2, \dots, x_9; c^j; 1; t_1^j, t_2^j, \dots, t_9^j\rangle \right] \quad (5.9)$$

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

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:

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

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

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

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.

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

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.

TO BE DONE! Add table with the simulation results from Liqui| and discuss them here.

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 two qubits, the algorithm would require 6 qubits making an IBM QC implementation again impossible. This can be seen from the initial state $|\psi_0\rangle$ 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:

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

where

$$|\Psi_x\rangle = \sum_{i=1}^N x_i |i\rangle \quad |\Psi_{t^m}\rangle = \sum_{i=1}^N t_i^m |i\rangle \quad (5.15)$$

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

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 $|c^m(A \text{ or } B)\rangle$ 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,

$$\begin{aligned} |\psi_1\rangle &= (H \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1}) |\psi_0\rangle \\ &= \frac{1}{2\sqrt{M}} \sum_{m=1}^M (|0\rangle [|\Psi_x\rangle + |\Psi_{t^m}\rangle] + |1\rangle [|\Psi_x\rangle - |\Psi_{t^m}\rangle]) |c^m(A \text{ or } B)\rangle |m\rangle \end{aligned} \quad (5.17)$$

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:

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

Note that $\sum_{i=1}^N |x_i - t_i^m|^2$ is the squared Euclidean distance between the input vector and the m^{th} 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, $Prob(|a\rangle = |0\rangle)$ is a measure of how suitable the training vectors are to classify the new input.

After the successful CM, the state is proportional to:

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

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

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

Note that the probability of measuring class $|1\rangle$ (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 $|1\rangle$ 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 $|c\rangle$.

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

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 $|0\rangle$ 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.

5.2.1 Implementing 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.

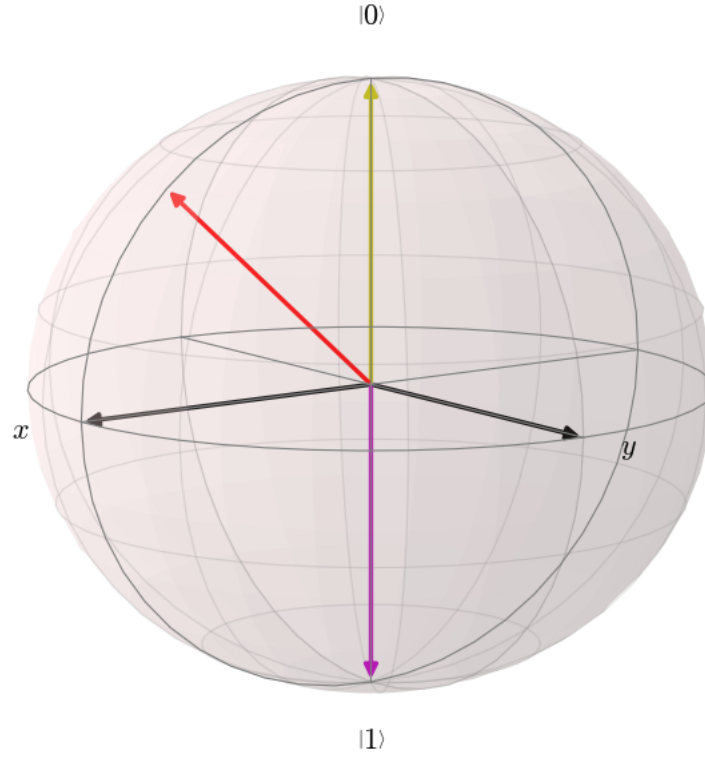


Figure 5.1: Simple binary classification problem of a quantum state

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$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	yellow ($ 0\rangle$)
$ 1\rangle$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	purple ($ 1\rangle$)

Table 5.5: Training set

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

Table 5.6: 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,

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

where in the case of the selected classification problem:

$$|\Psi_x\rangle = e^{-i\frac{\pi}{8}} [0.92388 |0\rangle + 0.38268 |1\rangle] \quad (5.22)$$

$$|\Psi_{t1}\rangle = |0\rangle \quad (5.23)$$

$$|\Psi_{t2}\rangle = |1\rangle \quad (5.24)$$

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 $1 \rightarrow 0$ and $2 \rightarrow 1$. 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 i^{th} 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,

$$|\chi_0\rangle = |a\rangle |d\rangle |c\rangle |m\rangle = |0\rangle |0\rangle |0\rangle |0\rangle \quad (5.25)$$

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:

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

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

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

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,

$$\begin{aligned} |\chi_3\rangle &= GS |\chi_2\rangle = \frac{1}{2} \sum_{m=0}^1 [|0\rangle |\Psi_x\rangle + |1\rangle |0\rangle] |0\rangle |m\rangle \\ &= \frac{1}{2} \sum_{m=0}^1 \left[|0\rangle e^{-i\frac{\pi}{8}} [0.92388 |0\rangle + 0.38268 |1\rangle] + |1\rangle |0\rangle \right] |0\rangle |m\rangle \end{aligned} \quad (5.28)$$

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 follows,

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

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

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

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 :

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

subbing into Equ. 5.28:

$$\begin{aligned} |\chi_3\rangle &= GS |\chi_2\rangle = (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})(CR_y(\frac{\pi}{4})(a, d) \otimes \mathbb{1} \otimes \mathbb{1}) |\chi_2\rangle \\ &= (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1}) \left[\frac{1}{2} \sum_{m=0}^1 [|0\rangle |0\rangle + |1\rangle |\Psi_x\rangle] |0\rangle |m\rangle \right] \\ &= \frac{1}{2} \sum_{m=0}^1 [|0\rangle |\Psi_x\rangle + |1\rangle |0\rangle] |0\rangle |m\rangle \end{aligned} \quad (5.32)$$

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:

$$\begin{aligned} |\chi_3\rangle &= \frac{1}{2} \left[[|0\rangle |\Psi_x\rangle + |1\rangle |0\rangle] |0\rangle |0\rangle + [|0\rangle |\Psi_x\rangle + |1\rangle |0\rangle] |0\rangle |1\rangle \right] \\ &= \frac{1}{2} \left[[|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^0}\rangle] |0\rangle |0\rangle + [|0\rangle |\Psi_x\rangle + |1\rangle |0\rangle] |0\rangle |1\rangle \right] \end{aligned} \quad (5.33)$$

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:

$$\begin{aligned} |\chi_4\rangle &= CCNOT(a, m, d) |\chi_3\rangle \\ &= \frac{1}{2} \left[[|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^0}\rangle] |0\rangle |0\rangle + [|0\rangle |\Psi_x\rangle + |1\rangle |1\rangle] |0\rangle |1\rangle \right] \\ &= \frac{1}{2} \left[[|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^0}\rangle] |0\rangle |0\rangle + [|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^1}\rangle] |0\rangle |1\rangle \right] \end{aligned} \quad (5.34)$$

The class qubit for the first training vector is already in the correct $|0\rangle$ state. 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,

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

and can be rewritten as:

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

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.

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:

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

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:

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

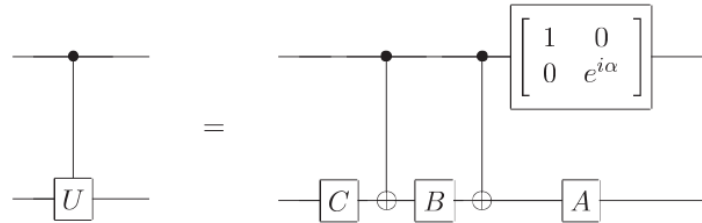


Figure 5.2: Circuit decomposition of controlled-U quantum gate.³

³Reprinted from Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000. Copyright 2010 by Nielsen & Chuang.

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:

$$ABC = \mathbb{1} \quad (5.39)$$

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:

$$e^{i\alpha}AXBXC = U \quad (5.40)$$

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

$$A = R_z(\beta)R_y\left(\frac{\gamma}{2}\right) \quad (5.41)$$

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

$$C = R_z\left(\frac{\delta - \beta}{2}\right) \quad (5.43)$$

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.

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

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 :

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

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:

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

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

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

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

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

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

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

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

$$(5.52)$$

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:

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

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

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

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

$$C = R_z\left(\frac{\delta - \beta}{2}\right) = R_z\left(-\frac{9}{16}\pi\right) = ? \quad (5.56)$$

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

Quantum gate A is just equal to the identity gate which certainly is 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 (?) . 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 defined in Section 2.1.1.

Theorem: Solovay-Kitaev theorem

Given a universal quantum set G of $SU(2)$ and a desired accuracy $\epsilon > 0$, 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 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).

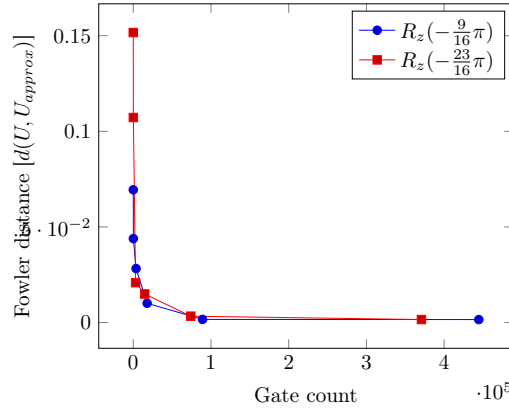


Figure 5.3: Fowler distance plotted against the number of required quantum gates

The Solovay-Kitaev theorem is so powerful 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, v0.03'¹⁰ developed in Python by Paul Pham was used to run the Solovay-Kitaev algorithm described by Dawson and Nielsen (2005).

To measure how good a particular gate approximation is compared to the desired gate W a new distance metric needs to be introduced.

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:

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

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} . It clearly shows an exponential increase in the number of gates for decreasing Fowler distance. For example, 109 gates suffice to achieve $d(W_{approx}, R_z(-\frac{23}{16}\pi)) = 0.10722$. However, to reduce the Fowler distance to $d(W_{approx}, R_z(-\frac{23}{16}\pi)) = 0.01494$, 14721 gates are required.

Fig. 5.5 is a visual aid to understanding Fowler distances. It shows five different vectors trying to approximate the action of the

¹⁰The software package "Quantum Compiler, v0.03" by Paul Pham can be downloaded from <https://sourceforge.net/projects/quantumcompiler/files/v0.03/>.

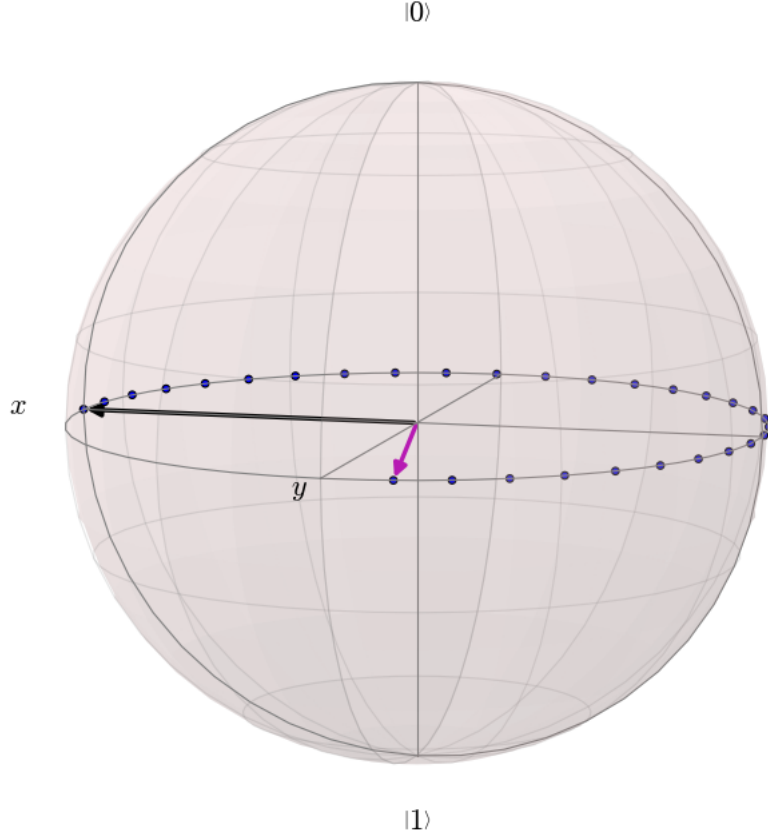


Figure 5.4: Action of $B = R_z(-\frac{23}{16}\pi)$ on the state $|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$

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

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 yields a Fowler distance of $d = 0.15165$. The resulting state vector is coloured green in Fig. 5.5 and is still far off from the desired vector \vec{b} in Fig. 5.4. With 109 gates, $B_{approx,2}$ lowers the Fowler distance to $d = 0.10722$ resulting in the blue vector still not being close to \vec{b} .

However, the yellow vector in Fig. 5.5 resulting from a gate sequence with

Using $B_{approx,3}$ the Fowler distance drops to $d = 0.02086$ leading to the yellow vector in Fig. 5.5. This vector can be considered a 'close' approximation of the desired state \vec{b} . However, $B_{approx,3}$ requires the implementation of 2997 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} that they are essentially the same. $B_{approx,4}$ needs 370,813 gates to approximate B to such a small Fowler distance.

Approx. Gate	Fowler distance	Gate count	Execution time
$R_z(-\frac{23}{16}\pi)$	0.15165	25	$\sim 3\text{ }\mu\text{s}$
	0.10722	109	$\sim 14\text{ }\mu\text{s}$
	0.02086	2997	$\sim 390\text{ }\mu\text{s}$
	0.01494	14721	$\sim 1914\text{ }\mu\text{s}$
	0.003327	74009	$\sim 9621\text{ }\mu\text{s}$
	0.001578	370813	$\sim 48\text{ }206\text{ }\mu\text{s}$

Table 5.7: SK algorithm results

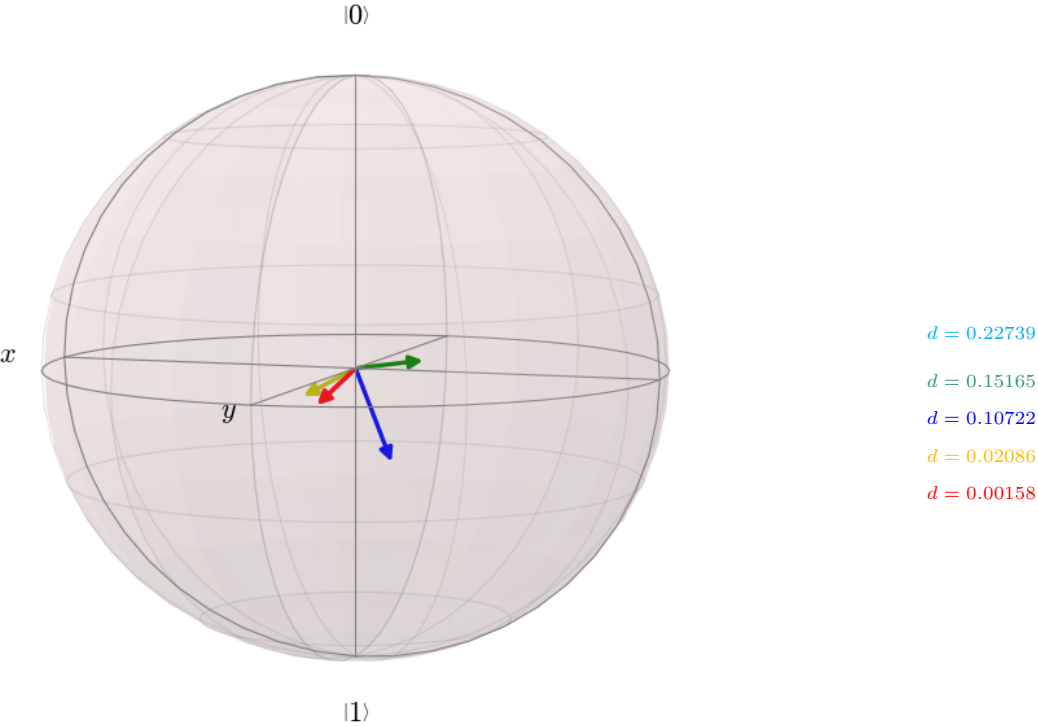


Figure 5.5: Various Fowler distances visualized on Bloch sphere

Adding complexities

Executing the SK algorithm adds to our overall algorithmic complexity:

Complexity analysis

The akNN algorithm itself was found to run in constant time $\mathcal{O}(\frac{1}{\text{Prob}(CM)})$. 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:

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

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

In conclusion, solely due to state preparation the initial constant complexity of

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

increased to polylogarithmic complexity of

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

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

It has been shown that a series of decompositions are required to compile the quantum state preparation for the aKNN algorithm into a series of quantum gates consisting only of gates from the available gate set. In this particular classification problem the constant complexity of the aKNN algorithm is dominated by the complexity of the quantum compiling.

STEPPING UP THE GAME SINCE IBM DOESN'T WORK SO FAR

5.2.2 Simulating the amplitude-based kNN algorithm

Diffusion matrix from quantum random walks

HELLINGER DISTANCE!

Initialization of amplitude distribution is non-trivial

Need for a simpler way - using diffusion matrix from quantum random walks

For the simplest case, consider the entries in the classical probability vector v to be normal distributed, e.g.

CHECK THE ORDER!

$$v = \begin{pmatrix} 0.064180 \\ 0.146860 \\ 0.146770 \\ 0.341590 \\ 0.026840 \\ 0.063590 \\ 0.061700 \\ 0.148470 \end{pmatrix} \quad (5.62)$$

The goal is to encode this classical data into a quantum memory state $|M\rangle$ of the form,

$$\begin{aligned} |M\rangle = & 0.064180 |000\rangle + 0.146860 |100\rangle + 0.146770 |010\rangle + 0.341590 |001\rangle \\ & + 0.026840 |110\rangle + 0.063590 |011\rangle + 0.061700 |101\rangle + 0.148470 |111\rangle \end{aligned} \quad (5.63)$$

Furthermore, a useful tool of visualizing the HDs between binary patterns made from three qubits is a 3-D cube as shown in Fig. 5.6. On the cube, adjacent qubit patterns have a HD of 1 and the HD increases by 1 for every additional corner. For example, the qubit state $|000\rangle$ is adjacent to $|100\rangle$ since they only differ in one qubit ($HD = 1$). Moving one more corner yields the state $|101\rangle$ or $|110\rangle$ which both have a HD of 2 compared to $|000\rangle$.

This way of visualizing HDs can be extended to the 16 binary patterns made by four qubits that can be visualized on a 4-D cube, also called tesseract, as illustrated in Fig. 5.7.

The idea of a coin operator C can be borrowed from the theory of quantum random walks to initialize a gaussian distribution centered around a chosen binary qubit pattern. For this purpose, the coin operator is defined as,

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

where $0 \leq \delta \leq 1$.

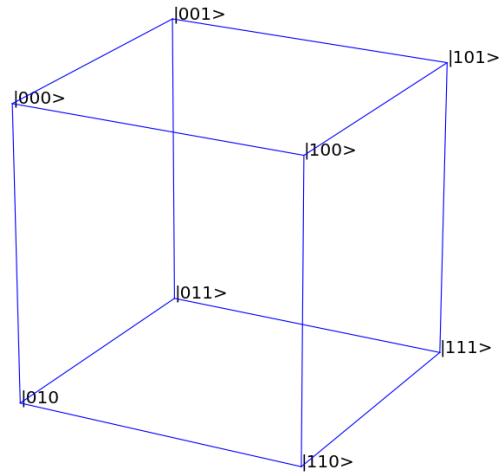


Figure 5.6: Visualizing Hamming distances on a 3-D cube

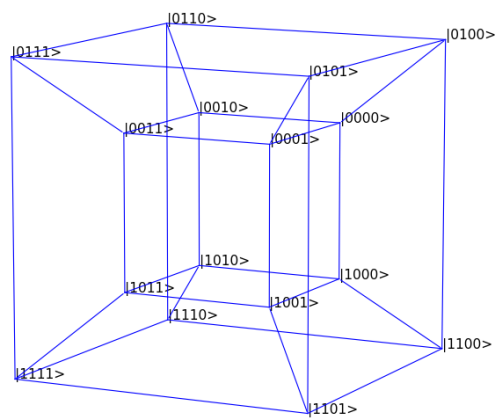


Figure 5.7: Visualizing Hamming distances on a 4-D cube (tesseract)

Chapter 6

Outlook

Chapter 7

Conclusion

Chapter 8

Personal reflection

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