Maastricht University

UNIVERSITY OF KWAZULU-NATAL
CENTRE FOR QUANTUM TECHNOLOGY

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

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

Indicies

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

Results and Discussion

Some more general text Why are some parts simulating and some parts using IBM? All the code can be found on GitHub[7]

1.1 Simulating the qubit-based kNN algorithm

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

$$b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^2 \text{ where } b_0, b_1, b_2 \in \{0, 1\} \tag{1.1}$$

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 1.1. 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 1.1. 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.

[7]Link 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 1.1: Training data set I

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

Table 1.2: Input data set I

In the more difficult evaluation stage, two more blue and red colours are added to training data set I resulting in training data set II shown in Table 1.1. 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 1.1. 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 1.3: Training data set II

Colour	Binary 9-bit RGB string	Expected class
🟥	100 000 000	red ($\|0\rangle$)
🟧	110 100 000	red ($\|0\rangle$)
🟥	101 100 100	red ($\|0\rangle$)
🟥	110 100 110	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 1.4: Input data set II

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

$$|T\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{N} |t_1^j, t_2^j, ...t_9^j; c^j\rangle \tag{1.2}$$

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

$$|\phi_0\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |t_1^6, t_2^6, ..., t_9^6; u_1 = 0, u_2 = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{1.3}$$

Equ. 1.3 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. 1.2.

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

$$|\psi_0\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |x_1, x_2, ...x_9; t_1^j, t_2^j, ...t_9^j; c^j; 0\rangle \tag{1.4}$$

The trick now is to realize that Equ. 1.3 and 1.4 contain the same number of qubits. Firstly, each of them has nine qubits in the first register. Secondly, Equ. 1.3 has two utility qubits that are balanced by the ancilla and the class qubit in Equ. 1.4. Lastly, there are again nine qubits in the third register in Equ. 1.3 and the second register in Equ. 1.4. Thus, $|\psi_0\rangle$ and $|\phi\rangle$ both contain $9+9+2 = 20$ qubits. Since $|\phi\rangle$ is the current state of the qubits in the simulation, one simply needs to redefine the utility qubits such that the first one becomes the class qubit $|u_1\rangle = |c^j\rangle$ and the second utility qubit becomes the ancilla $|u_2\rangle = |a\rangle$. Note that class and ancilla qubit are currently still in the $|0\rangle$ state as indicated in the current quantum state $|\phi_1\rangle$:

$$|\phi_1\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |t_1^6, t_2^6, ..., t_9^6; c^j = 0; a = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{1.5}$$

Equ. 1.5 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,

$$|\phi_2\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |x_1, x_2, ..., x_9; c^j = 0; a = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{1.6}$$

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

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

$$|\phi_3\rangle = \frac{1}{\sqrt{6}} \sum_{j=1}^{6} |x_1, x_2, ..., x_9; c^j; a = 0; t_1^j, t_2^j, ..., t_9^j\rangle \tag{1.7}$$

Note that the class qubit is now not in the $|0\rangle$ state only. Inspection of Equ. 1.7 reveals that $|\phi_3\rangle$ is identical to the desired initial state $|\psi_0\rangle$ defined in Equ. 1.4 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.

$$|\phi_4\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \Big[|x_1, x_2, ..., x_9; c^j; 0; t_1^j, t_2^j, ..., t_9^j\rangle + |x_1, x_2, ..., x_9; c^j; 1; t_1^j, t_2^j, ..., t_9^j\rangle \Big] \tag{1.8}$$

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

$$\begin{aligned} |\phi_5\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \prod_{s=1}^{9} X(t_s^j) CNOT(x_s, t_s^j) \Big[& |x_1, x_2, ..., x_9; c^j; 0; t_1^j, t_2^j, ..., t_9^j\rangle \\ & + |x_1, x_2, ..., x_9; c^j; 1; t_1^j, t_2^j, ..., t_9^j\rangle \Big] \end{aligned} \tag{1.9}$$

$$= \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \Big[|x_1, x_2, ..., x_9; c^j; 0; d_1^j, d_2^j, ...d_9^j\rangle + |x_1, x_2, ..., x_9; c^j; 1; d_1^j, d_2^j, ...d_9^j\rangle \Big] \tag{1.10}$$

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. **??** and **??** with $n = 9$ in the case of 9-bit RGB classification. According to Trugenberger (2001) the operator U can be decomposed as follows:

$$U |\phi_5\rangle = e^{-i\frac{\pi}{2n}K} |\phi_5\rangle = e^{-i\frac{\pi}{18}K} |\phi_5\rangle = \prod_{f=1}^{9} CL^{-2}(a, t_f) \prod_{k=1}^{9} L(t_k) |\phi_5\rangle \tag{1.11}$$

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

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

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

Table 1.5: Classification results after 100 runs using input data set I. Theoretical predictions (marked with asterisks) on top, experimental results at the bottom.

$$|\phi_6\rangle = U\,|\phi_5\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \Big[e^{i\frac{\pi}{18} d_H(\vec{x},\vec{v}^p)}\, |x_1, x_2, ..., x_9; c^j; 0; d_1^j, d_2^j, ...d_9^j\rangle + e^{-i\frac{\pi}{18} d_H(\vec{x},\vec{v}^p)}\, |x_1, x_2, ..., x_9; c^j; 1; d_1^j, d_2^j, ...d_9^j\rangle \Big] \tag{1.12}$$

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.

$$|\phi_7\rangle = (\mathbb{1} \otimes \mathbb{1} \otimes H \otimes \mathbb{1})\,|\phi_6\rangle = \frac{1}{\sqrt{12}} \sum_{j=1}^{6} \Big[cos\big[\frac{\pi}{18} d_H(\vec{x},\vec{t^j})\big]\, |x_1, x_2, ..., x_9; c^j; 0; d_1^j, d_2^j, ...d_9^j\rangle + sin\big[\frac{\pi}{18} d_H(\vec{x},\vec{t^j})\big]\, |x_1, x_2, ..., x_9; c^j; 1; d_1^j, d_2^j, ...d_9^j\rangle \Big] \tag{1.13}$$

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$|\rangle$ 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. 1.4 that contains the input pattern (two qubits), the training pattern (two

Colour	Binary 9-bit RGB string	$Prob(CM)$	$Prob$ $(\|c\rangle = \|0\rangle)$	$Prob$ $(\|c\rangle = \|1\rangle)$	Expected class	Algorithm output
🟥	100 000 000	0.7543* / 0.8500	0.5515* / 0.5529	0.4485* / 0.4471	red ($\|0\rangle$)	red ($\|0\rangle$)
🟥	110 100 000	0.6312* / 0.6100	0.6710* / 0.7377	0.3289* / 0.2623	red ($\|0\rangle$)	red ($\|0\rangle$)
🟥	101 100 100	0.6996* / 0.7200	0.5821* / 0.5694	0.4179* / 0.4306	red ($\|0\rangle$)	red ($\|0\rangle$)
🟥	110 100 110	0.6470* / 0.6600	0.5229* / 0.5000	0.4771* / 0.5000	red ($\|0\rangle$)	blue ($\|1\rangle$)
🟦	000 000 110	0.6880* / 0.7000	0.3760* / 0.3714	0.6240* / 0.6286	blue ($\|1\rangle$)	blue ($\|1\rangle$)
🟦	000 100 111	0.5649* / 0.5500	0.2266* / 0.2182	0.7734* / 0.7818	blue ($\|1\rangle$)	blue ($\|1\rangle$)
🟦	100 100 111	0.6236* / 0.6100	0.3330* / 0.3279	0.6670* / 0.6721	blue ($\|1\rangle$)	blue ($\|1\rangle$)
🟪	110 100 101	0.6807* / 0.7000	0.4970* / 0.5857	0.5030* / 0.4143	blue ($\|1\rangle$)	red ($\|0\rangle$)

Table 1.6: Classification results after 100 runs using input data set II. Theoretical predictions (marked with asterisks) on top, experimental results at the bottom.

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.

1.2 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. **??** in Section **??**.

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

$$|\psi_0\rangle = \frac{1}{\sqrt{2M}} \sum_{m=1}^{M} (|0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^m}\rangle) |c^m(A \textit{ or } B)\rangle |m\rangle \tag{1.14}$$

where

$$|\Psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle \qquad |\Psi_{t^m}\rangle = \sum_{i=1}^{N} t_i^m |i\rangle \tag{1.15}$$

$$e.g. \quad \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \quad \rightarrow \quad |n\rangle = \sqrt{0.6}\,|0\rangle + \sqrt{0.4}\,|1\rangle \tag{1.16}$$

The first qubit in Equ. 1.14 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. **??**) as shown in Equ. 1.15. 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. 1.15). Lastly, there is the class qubit $|c^m(A \textit{ or } B)\rangle$ 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,

$$\begin{aligned} |\psi_1\rangle &= (H \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})\,|\psi_0\rangle \\ &= \frac{1}{2\sqrt{M}} \sum_{m=1}^{M} (|0\rangle[|\Psi_x\rangle + |\Psi_{t^m}\rangle] + |1\rangle[|\Psi_x\rangle - |\Psi_{t^m}\rangle])\,|c^m(A \textit{ or } B)\rangle\,|m\rangle \end{aligned} \tag{1.17}$$

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

$$Prob(CM) = Prob(|a\rangle = |0\rangle) = 1 - \frac{1}{4M} \sum_{m=1}^{M} \sum_{i=1}^{N} |\, x_i - t_i^m \,|^2 \tag{1.18}$$

Note that $\sum_{i=1}^{N} \mid x_i - t_i^m \mid^2$ is the squared Euclidean distance between the input vector and the m^{th} training vector. Equ. 1.18 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:

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^{M} \sum_{i=1}^{N} (x_i + t_i^m) |0\rangle |i\rangle |c^m(A \text{ } or \text{ } B)\rangle |m\rangle \tag{1.19}$$

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

$$Prob(|c^m\rangle = |1(B)\rangle) = \sum_{m|c^m=1(B)} 1 - \frac{1}{4M} \sum_{i=1}^{N} \mid x_i - t_i^m \mid^2 \tag{1.20}$$

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.

1.2.1 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.

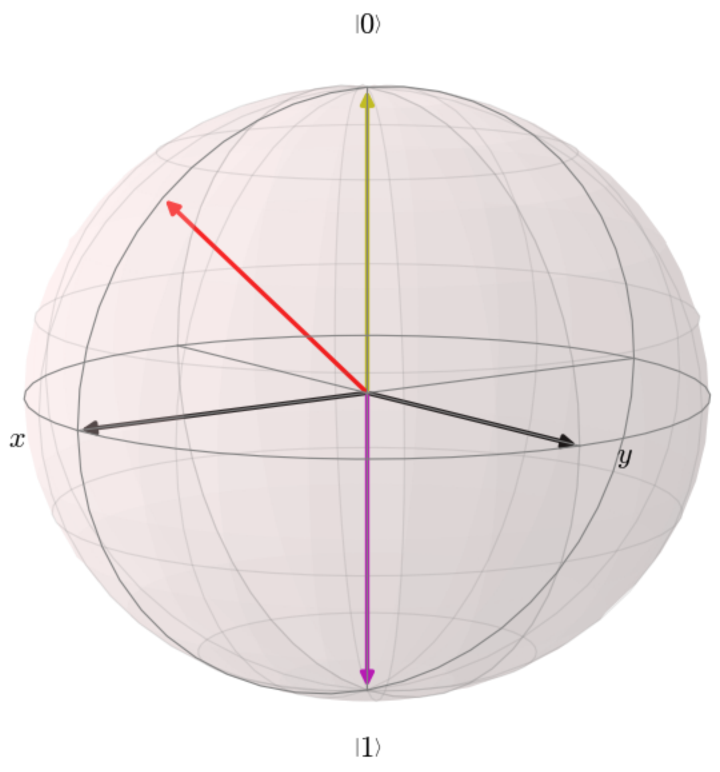

Figure 1.1: Simple binary classification problem of a quantum state

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 **??**). The training set is listed in Table 1.2.1 and consists only of the $|0\rangle$ and the $|1\rangle$ vector depicted in Fig. 1.1 as the yellow and purple vector respectively. The red vector in Fig. 1.1 will be the input vector and its qubit state and vector representation are given in Table 1.2.1.

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 1.7: Training set

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

Table 1.8: Input vector

Initial state preparation

The first step towards an actual implementation of this problem is to prepare the initial quantum state $|\psi_0\rangle$ previously defined in Equ. 1.14 to be of the form,

$$|\psi_0\rangle = \frac{1}{\sqrt{2M}} \sum_{m=1}^{M} \left(|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^m}\rangle\right)|c^m\rangle\,|m\rangle \tag{1.21}$$

where in the case of the selected classification problem:

$$|\Psi_x\rangle = e^{-i\frac{\pi}{8}}\left[0.92388\,|0\rangle + 0.38268\,|1\rangle\right] \tag{1.22}$$
$$|\Psi_{t^1}\rangle = |0\rangle \tag{1.23}$$
$$|\Psi_{t^2}\rangle = |1\rangle \tag{1.24}$$

Since there are only two training vectors the index m in Equ. 1.21 only takes the values 1 and 2. However, the m qubit can only take binary values such that we need to redefine $1 \to 0$ and $2 \to 1$. With this observation, the required number of qubits can be deduced from Equ. 1.21: 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,

$$|\chi_0\rangle = |a\rangle\,|d\rangle\,|c\rangle\,|m\rangle = |0\rangle\,|0\rangle\,|0\rangle\,|0\rangle \tag{1.25}$$

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

$$|\chi_1\rangle = (\mathbb{1}\otimes\mathbb{1}\otimes\mathbb{1}\otimes H)\,|0\rangle\,|0\rangle\,|0\rangle\,|0\rangle = \frac{1}{\sqrt{2}}\sum_{m=0}^{1}|0\rangle\,|0\rangle\,|0\rangle\,|m\rangle \tag{1.26}$$

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

$$|\chi_2\rangle = (H\otimes\mathbb{1}\otimes\mathbb{1}\otimes\mathbb{1})\,|\chi_1\rangle = \frac{1}{2}\sum_{m=0}^{1}|0\rangle\,|0\rangle\,|0\rangle\,|m\rangle + |1\rangle\,|0\rangle\,|0\rangle\,|m\rangle = \frac{1}{2}\sum_{m=0}^{1}\left[\,|0\rangle\,|0\rangle + |1\rangle\,|0\rangle\,\right]|0\rangle\,|m\rangle \tag{1.27}$$

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,

$$\begin{aligned}|\chi_3\rangle = GS\,|\chi_2\rangle &= \frac{1}{2}\sum_{m=0}^{1}\left[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|0\rangle\,\right]|0\rangle\,|m\rangle \\ &= \frac{1}{2}\sum_{m=0}^{1}\Big[\,|0\rangle\, e^{-i\frac{\pi}{8}}\left[0.92388\,|0\rangle + 0.38268\,|1\rangle\,\right] + |1\rangle\,|0\rangle\Big]\,|0\rangle\,|m\rangle\end{aligned} \tag{1.28}$$

By looking closely at Fig. 1.1 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,

$$R_y(\vartheta) = e^{-i\vartheta\frac{Y}{2}} = \cos\frac{\vartheta}{2}\mathbb{1} - i\sin\frac{\vartheta}{2}Y = \begin{pmatrix}\cos\frac{\vartheta}{2} & -\sin\frac{\vartheta}{2}\\ \sin\frac{\vartheta}{2} & \cos\frac{\vartheta}{2}\end{pmatrix} \tag{1.29}$$

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

$$(\mathbb{1} \otimes R_y(\frac{\pi}{4}) \otimes \mathbb{1} \otimes \mathbb{1}) |\chi_2\rangle = \frac{1}{2} \sum_{m=0}^{1} \big[\, |0\rangle |\Psi_x\rangle + |1\rangle |\Psi_x\rangle \,\big] |0\rangle |m\rangle \tag{1.30}$$

This, however, is not the desired state $|\chi_3\rangle$ defined in Equ. 1.28 since the input vector $|\Psi_x\rangle$ should only be entangled with the $|0\rangle$ state of the ancilla. To achieve this type of entanglement the controlled version of the y-rotation gate, $CR_y(\frac{\pi}{4})(c,t)$, is required. When applying it using the a qubit as control and the d qubit as target the input vector will be entangled with the $|1\rangle$ state of the ancilla. Flipping the a qubit with an X gate moves the input vector to the $|0\rangle$ state of the ancilla. Hence, the desired state $|\chi_3\rangle$ is obtained by applying the following gate sequence GS:

$$GS = (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})(CR_y(\frac{\pi}{4})(a,d) \otimes \mathbb{1} \otimes \mathbb{1}) \tag{1.31}$$

subbing into Equ. 1.28:

$$\begin{aligned} |\chi_3\rangle &= GS\,|\chi_2\rangle = (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1})(CR_y(\frac{\pi}{4})(a,d) \otimes \mathbb{1} \otimes \mathbb{1}) |\chi_2\rangle \\ &= (X \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1}) \Big[\frac{1}{2} \sum_{m=0}^{1} \big[\, |0\rangle |0\rangle + |1\rangle |\Psi_x\rangle \,\big] |0\rangle |m\rangle \Big] \\ &= \frac{1}{2} \sum_{m=0}^{1} \Big[\, |0\rangle |\Psi_x\rangle + |1\rangle |0\rangle \,\Big] |0\rangle |m\rangle \end{aligned} \tag{1.32}$$

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. 1.22 were redefined to $|\Psi_{t^0}\rangle$ and $|\Psi_{t^1}\rangle$ respectively. Expanding the sum in Equ. 1.32 demonstrates that $|\Psi_{t^0}\rangle = |0\rangle$ is already at its desired place:

$$\begin{aligned} |\chi_3\rangle &= \frac{1}{2} \Big[\big[\, |0\rangle |\Psi_x\rangle + |1\rangle |0\rangle \,\big] |0\rangle |0\rangle + \big[\, |0\rangle |\Psi_x\rangle + |1\rangle |0\rangle \,\big] |0\rangle |1\rangle \Big] \\ &= \frac{1}{2} \Big[\big[\, |0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^0}\rangle \,\big] |0\rangle |0\rangle + \big[\, |0\rangle |\Psi_x\rangle + |1\rangle |0\rangle \,\big] |0\rangle |1\rangle \Big] \end{aligned} \tag{1.33}$$

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:

$$\begin{aligned} |\chi_4\rangle &= CCNOT(a,m,d)\,|\chi_3\rangle \\ &= \frac{1}{2} \Big[\big[\, |0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^0}\rangle \,\big] |0\rangle |0\rangle + \big[\, |0\rangle |\Psi_x\rangle + |1\rangle |1\rangle \,\big] |0\rangle |1\rangle \Big] \\ &= \frac{1}{2} \Big[\big[\, |0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^0}\rangle \,\big] |0\rangle |0\rangle + \big[\, |0\rangle |\Psi_x\rangle + |1\rangle |\Psi_{t^1}\rangle \,\big] |0\rangle |1\rangle \Big] \end{aligned} \tag{1.34}$$

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

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

$$
\begin{aligned}
|\chi_5\rangle &= CNOT(d,c)\,|\chi_4\rangle \\
&= \frac{1}{2}\Big[\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big]\,|0\rangle\,|0\rangle + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^1}\rangle\,\big]\,|1\rangle\,|1\rangle\Big] \qquad (1.35)
\end{aligned}
$$

and can be rewritten as:

$$
\begin{aligned}
|\chi_5\rangle &= \frac{1}{2}\sum_{m=1}^{2}\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^0}\rangle\,\big] + \big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^1}\rangle\,\big]\,|c^m\rangle\,|m\rangle \\
&= \frac{1}{2}\sum_{m=1}^{2}\big[\,|0\rangle\,|\Psi_x\rangle + |1\rangle\,|\Psi_{t^m}\rangle\,\big]\,|c^m\rangle\,|m\rangle \qquad (1.36)
\end{aligned}
$$

When comparing Equ. 1.36 to $|\psi_0\rangle$ in Equ. 1.14 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:

$$
CU = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & U \end{pmatrix} \qquad (1.37)
$$

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. 1.2 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:

$$
P = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} \qquad (1.38)
$$

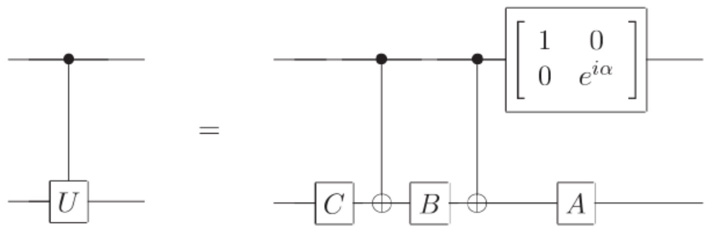

Figure 1.2: Circuit decomposition of controlled-U quantum gate.[3]

[3]Reprinted 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. 1.2) is $|0\rangle$ the gate combination ABC is applied to the target qubit (bottom qubit in Fig. 1.2) and has to equal the identity gate:

$$ABC = \mathbb{1} \tag{1.39}$$

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 satified:

$$e^{i\alpha}AXBXC = U \tag{1.40}$$

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

$$A = R_z(\beta)R_y(\frac{\gamma}{2}) \tag{1.41}$$

$$B = R_y(-\frac{\gamma}{2})R_z(-\frac{\delta+\beta}{2}) \tag{1.42}$$

$$C = R_z(\frac{\delta-\beta}{2}) \tag{1.43}$$

where $R_z(\varrho)$ is the general rotation gate about the z-axis of the Bloch sphere. Similary to the $R_y(\vartheta)$ gate it can be obtained by exponentiating the Z gate as shown below in Equ. 1.44.

$$R_z(\varrho) = e^{-i\varrho\frac{Z}{2}} = \cos\frac{\varrho}{2}\mathbb{1} - i\sin\frac{\varrho}{2}Z = \begin{pmatrix} e^{-i\frac{\varrho}{2}} & 0 \\ 0 & e^{i\frac{\varrho}{2}} \end{pmatrix} \tag{1.44}$$

When subbing the expressions for A, B, C from Equ. 1.41 into Equ. 1.39 one will indeed obtain the identity operation (proof omitted). These choices of A, B, C are also a solution to Equ. 1.40. Subbing into Equ. 1.40 leads to the following expression for matrix U:

$$U = \begin{pmatrix} e^{i(\alpha-\frac{\beta}{2}-\frac{\delta}{2})}\cos\frac{\gamma}{2} & -e^{i(\alpha-\frac{\beta}{2}+\frac{\delta}{2})}\sin\frac{\gamma}{2} \\ e^{i(\alpha+\frac{\beta}{2}-\frac{\delta}{2})}\sin\frac{\gamma}{2} & e^{i(\alpha+\frac{\beta}{2}+\frac{\delta}{2})}\cos\frac{\gamma}{2} \end{pmatrix} \tag{1.45}$$

To decompose $CR_y(\frac{\pi}{4})$, one simply chooses $U = R_y(\frac{\pi}{4})$. Subbing $\vartheta = \frac{\pi}{4}$ into Equ. 1.29 the matrix representation of $R_y(\frac{\pi}{4})$ is obtained:

$$U = R_y(\frac{\pi}{4}) = \begin{pmatrix} \cos\frac{\pi}{8} & -\sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & \cos\frac{\pi}{8} \end{pmatrix} \tag{1.46}$$

Subbing this expression for U into Equ. 1.45 leads to:

$$U = \begin{pmatrix} \cos\frac{\pi}{8} & -\sin\frac{\pi}{8} \\ \sin\frac{\pi}{8} & \cos\frac{\pi}{8} \end{pmatrix} = \begin{pmatrix} e^{i(\alpha-\frac{\beta}{2}-\frac{\delta}{2})}\cos\frac{\gamma}{2} & -e^{i(\alpha-\frac{\beta}{2}+\frac{\delta}{2})}\sin\frac{\gamma}{2} \\ e^{i(\alpha+\frac{\beta}{2}-\frac{\delta}{2})}\sin\frac{\gamma}{2} & e^{i(\alpha+\frac{\beta}{2}+\frac{\delta}{2})}\cos\frac{\gamma}{2} \end{pmatrix} \tag{1.47}$$

When setting the expressions for the respective matrix entries in Equ. 1.47 equal the following system of non-linear complex equations is obtained :

$$\cos\frac{\pi}{8} = e^{i(\alpha-\frac{\beta}{2}-\frac{\delta}{2})}\cos\frac{\gamma}{2} \tag{1.48}$$
$$-\sin\frac{\pi}{8} = -e^{i(\alpha-\frac{\beta}{2}+\frac{\delta}{2})}\sin\frac{\gamma}{2} \tag{1.49}$$
$$\sin\frac{\pi}{8} = e^{i(\alpha+\frac{\beta}{2}-\frac{\delta}{2})}\sin\frac{\gamma}{2} \tag{1.50}$$
$$\cos\frac{\pi}{8} = e^{i(\alpha+\frac{\beta}{2}+\frac{\delta}{2})}\cos\frac{\gamma}{2} \tag{1.51}$$
$$\tag{1.52}$$

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

$$\alpha = \pi; \quad \beta = 2\pi; \quad \delta = \frac{7}{8}\pi; \quad \gamma = 0 \tag{1.53}$$

Subbing these parameters into the expressions for A, B and C defined in Equ. 1.41 yields,

$$A = R_z(\beta)R_y(\frac{\gamma}{2}) = R_z(2\pi) = \mathbb{1} \tag{1.54}$$
$$B = R_y(-\frac{\gamma}{2})R_z(-\frac{\delta+\beta}{2}) = R_z(-\frac{23}{16}\pi) = ? \tag{1.55}$$
$$C = R_z(\frac{\delta-\beta}{2}) = R_z(-\frac{9}{16}\pi) = ? \tag{1.56}$$
$$P = \begin{pmatrix}1 & 0\\ 0 & e^{i\alpha}\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi}\end{pmatrix} = Z \tag{1.57}$$

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. 1.55 and 1.56.

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 previously defined in Section **??**.

Theorem: Solovay-Kitaev theorem

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

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

The Solovay-Kitaev theorem is 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). This software package compares different gate approximations to the desired gate by a new distance metric called Fowler distance.

Definition: Fowler distance

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

$$d(W, W_{approx}) = \sqrt{\frac{2- \mid tr(W \cdot W^{\dagger}_{approx}) \mid}{2}} \tag{1.58}$$

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. 1.3 shows a plot visualizing how the Fowler distance depends on the number of gates in the approximating gate sequence W_{approx}. The number of gates in the approximating gate sequence is also called *gate count*. The plot clearly shows an exponential increase in the gate count for decreasing Fowler distance. For example, 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. BAD LOOKS! Update with 9/16 pi examples!

Fowler distance is best understood with a visual example using the gate B. 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. 1.4) results in the purple Bloch vector $\vec{b}$ shown in Fig. 1.4.

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

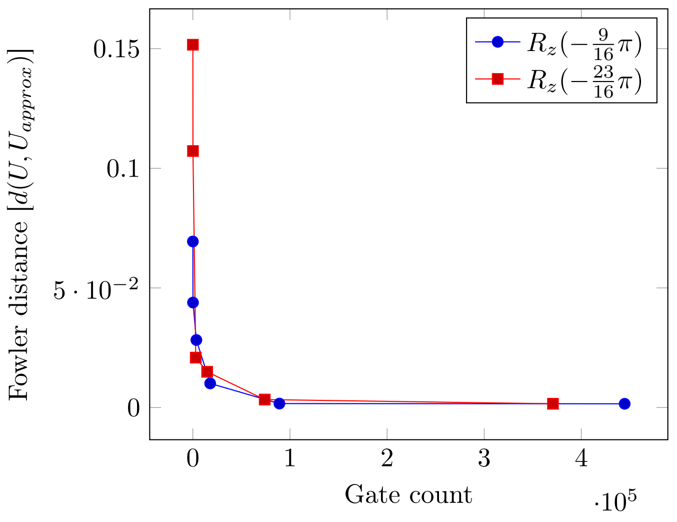

Figure 1.3: Fowler distance plotted against the required gate count for quantum gates $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$

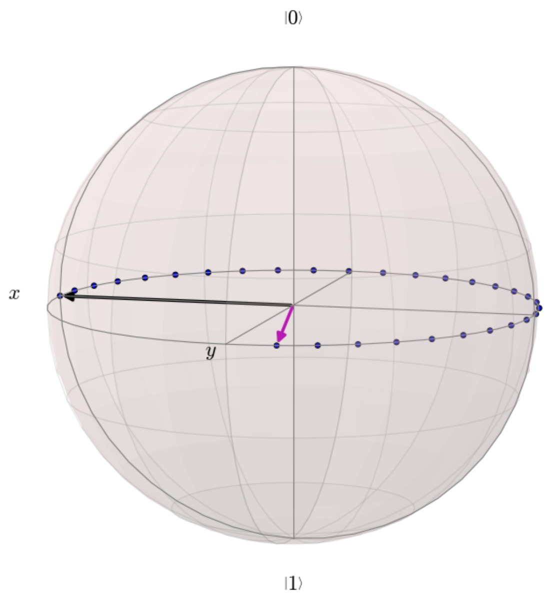

Figure 1.4: Action of $B = R_z(-\frac{23}{16}\pi)$ on the state $|\psi\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$

Fig. 1.5 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. 1.5. The green vector lies above the Bloch equator and is relatively far away from the desired vector $\vec{b}$ in Fig. 1.4. With 109 gates, $B_{approx,2}$ lowers the Fowler distance to $d = 0.10722$ resulting in the blue vector. It is also found below the Bloch equator and is still relatively far from $\vec{b}$. Using $B_{approx,3}$ the Fowler distance drops to $d = 0.02086$ leading to the yellow vector in Fig. 1.5. This vector is almost on the Bloch equator and relatively close to $\vec{b}$. Therefore, $B_{approx,3}$ can be considered a good approximation of B. However, $B_{approx,3}$ already requires the implementation of 2,997 single-qubit gates. The purple vector from Fig. 1.4 is also plotted in Fig. 1.5 but is not visible since the red vector with $d = 0.00158$ constitutes such a good approximation to the desired state $\vec{b}$. The red vector is the result of the action of $B_{approx,4}$ which, unfortunately, needs 370,813 gates to approximate B to such a small Fowler distance. Gate B was used as an example but such an exponential increase in the gate count is observed for any quantum gate W when applying the Solovay-Kitaev algorithm (CITATION).

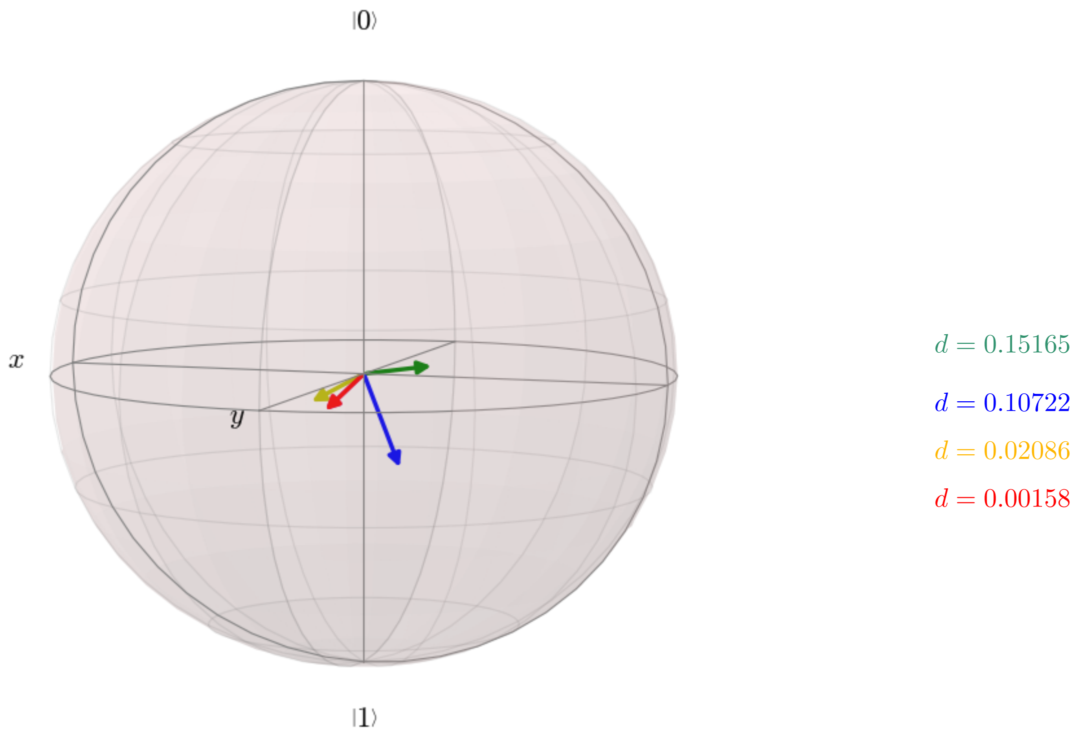

Figure 1.5: Various Fowler distances visualized on Bloch sphere

Implementing any quantum gate D inevitable takes some time t since it involves the use of hardware components e.g. targeting a specific electron with a laser pulse. Based on IBM's single-qubit and CNOT gate times described in Section ?? the total execution time of twelve different gate sequences, approximating $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$ to different Fowler distances, were calculated. The results can be seen in Table 1.9. Keeping in mind that in the best case, the maximum decoherence time of a qubit in IBM's QC is 112.4 µs most gate sequences are too long for an IBM implementation (coloured red in Table 1.9). Feasible execution times are marked green in Table 1.9. However, decoherence is not the only limiting factor since IBM only allows for 39 gates and one measurement gate. Thus, the only possibility would be to use a sequence of 25 gates for B and a sequence of 16 gates for C at the cost of relatively large Fowler distances. This, however, sums up to 41 gates which is still two gates more than the allowed gate count of the IBM Quantum Composer.

Approx. Gate	Fowler distance	Gate count	Execution time
$R_z(-\frac{23}{16}\pi)$	0.15165	25	~3 µs
	0.10722	109	~14 µs
	0.02086	2,997	~390 µs
	0.01494	14,721	~1914 µs
	0.003327	74,009	~9621 µs
	0.001578	370,813	~48 206 µs
$R_z(-\frac{9}{16}\pi)$	0.28390	16	~2 µs
	0.04389	146	~18 µs
	0.049511	728	~87 µs
	0.02823	3,622	~435 µs
	0.01008	17,838	~2141 µs
	0.00156	444,646	~53 358 µs

Table 1.9: Results of the Solovay-Kitaev algorithm for decomposition of $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$

Toffoli Gate Decomposition

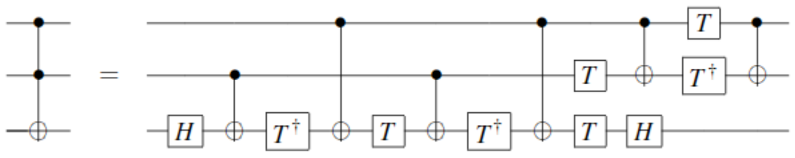

Figure 1.6: Decomposition of a Toffoli gate into six CNOT and nine single-qubit gates[12]

CANNOT BE IMPLEMENTED ON IBM QUANTUM COMPUTER

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

Complexity analysis

The akNN algorithm itself was found to run in constant time $\mathcal{O}(\frac{1}{Prob(CM)})$. However, to determine the overall algorithmic complexity all steps required to prepare the initial quantum state $|\psi_0\rangle$ from Equ. 1.14 need to be included. The decomposition of the $CR_y(\frac{\pi}{4})$ gate involved solving a system of non-linear equations by means of a root-finding algorithm. Any iterative root-finding algorithm e.g. Secant or Newton's method has a complexity of $\mathcal{O}(k)$ where k is the number of root finding iterations. Furthermore, the Solovay-Kitaev algorithm was required to find two single-qubit gate sequences approximating the two gates $B = R_z(-\frac{23}{16}\pi)$ and $C = R_z(-\frac{9}{16}\pi)$. According to Dawson and Nielsen (2005) the Solovay-Kitaev algorithm has a complexity of $\mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon}))$ for ϵ-approximations of m gates. Thus the total complexity is given by:

$$\mathcal{O}(\frac{1}{Prob(CM)}) + \mathcal{O}(k) + \mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon})) \tag{1.59}$$

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

$$\mathcal{O}(\frac{1}{Prob(CM)}) \tag{1.60}$$

increased to polylogarithmic complexity of

$$\mathcal{O}(m * log^{2.71}(\frac{m}{\epsilon})) \tag{1.61}$$

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

1.2.2 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!

$$v = \begin{pmatrix} 0.064180 \\ 0.146860 \\ 0.146770 \\ 0.341590 \\ 0.026840 \\ 0.063590 \\ 0.061700 \\ 0.148470 \end{pmatrix} \tag{1.62}$$

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

$$\begin{aligned} |M\rangle = \quad & 0.064180\,|000\rangle + 0.146860\,|100\rangle + 0.146770\,|010\rangle + 0.341590\,|001\rangle \\ & + 0.026840\,|110\rangle + 0.063590\,|011\rangle + 0.061700\,|101\rangle + 0.148470\,|111\rangle \end{aligned} \tag{1.63}$$

Furthermore, a useful tool of visualizing the HDs between binary patterns made from three qubits is a 3-D cube as shown in Fig. 1.7. 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. 1.8.

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,

$$C = \begin{pmatrix} \sqrt{\delta} & 1-\sqrt{\delta} \\ 1-\sqrt{\delta} & -\sqrt{\delta} \end{pmatrix} \tag{1.64}$$

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

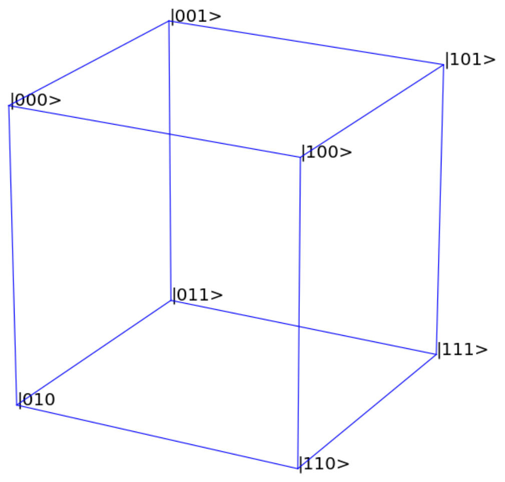

Figure 1.7: Visualizing Hamming distances on a 3-D cube

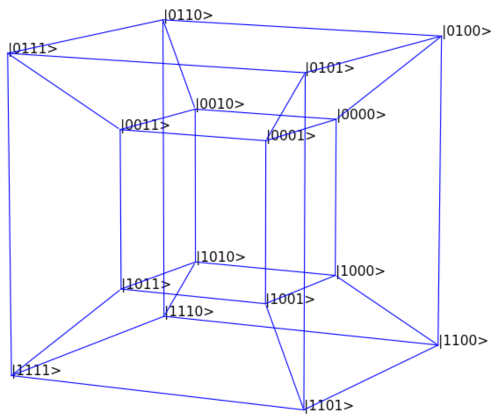

Figure 1.8: Visualizing Hamming distances on a 4-D cube (tesseract)

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