

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	7
2.2.1 Single Qubit Gates	8
2.2.2 Multiple Qubit Gates	10
2.3 Classical Machine Learning	11
2.3.1 k-nearest Neighbour Algorithm	11
3 Methods	15
3.1 Liqui $ \rangle$	15
3.2 IBM Quantum Experience	16
4 Literature Review: Quantum-enhanced Machine Learning	17
4.0.1 Quantum State Preparation Algorithms	17
4.0.2 Quantum k-nearest Neighbour Algorithm	19
5 Results and Discussion	21
6 Outlook	23
7 Conclusion	25
8 Personal reflection	27
References	28

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

Indicies

a	Ambient
air	Air

Acronyms and Abbreviations

QML	Quantum machine learning
QC	Quantum computer
Prob	Probability
PM	Post-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, 2016), 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, & 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!

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.

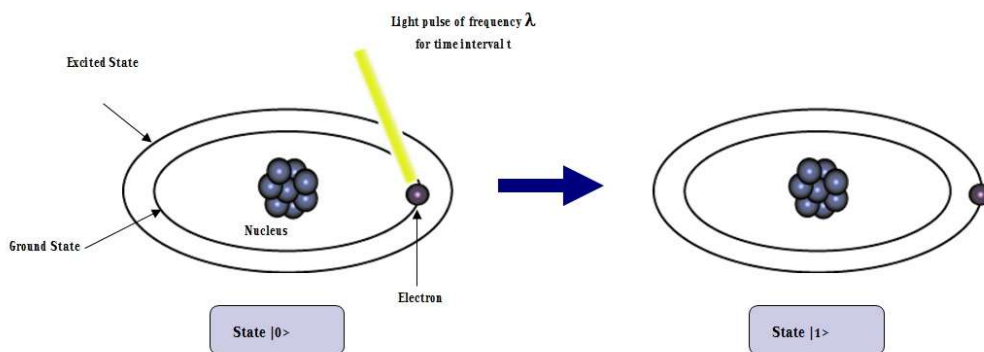


Figure 2.1: A simple example of a physical qubit using the electron of a hydrogen atom

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

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.

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 phase factors or amplitudes. $|0\rangle$ 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!

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (2.2)$$

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

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

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

$$Prob(|0\rangle) = |\alpha|^2 \quad (2.4)$$

and $|1\rangle$ with a probability of

$$Prob(|1\rangle) = |\beta|^2 \quad (2.5)$$

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:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2.6)$$

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.

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:

$$U|\psi\rangle = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a * \alpha + b * \beta \\ c * \alpha + d * \beta \end{pmatrix} \quad (2.7)$$

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. 2.8 and 2.9). All quantum logic gates must be unitary since this preserves the normalization of the qubit state it is acting on.

$$a) | \det(U) | = 1 \quad (2.8)$$

$$b) UU^\dagger = U^\dagger U = \mathbb{1} = UU^{-1} = U^{-1}U \quad (2.9)$$

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:

$$|q\rangle = \cos \frac{\theta}{2} |0\rangle + e^{i\phi} \sin \frac{\theta}{2} |1\rangle \quad (2.10)$$

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.

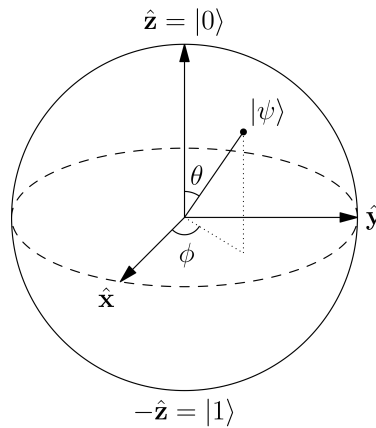


Figure 2.2: An arbitrary two-dimensional qubit $|\psi\rangle$ visualized on the Bloch sphere.²

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.

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:

$$|\psi\rangle \otimes |\psi\rangle = |0\rangle \otimes |0\rangle = |00\rangle \quad (2.11)$$

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:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \text{red} * \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \text{green} * \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad (2.12)$$

The last expression in Equ. 2.12 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. 2.7) which demands for a new gate formalism for multi qubit systems.

Consider wanting to apply an arbitrary single-qubit gate U (Equ. 2.7) 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,

$$U \otimes \mathbb{1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & b * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ c * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & d * \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{pmatrix} \quad (2.13)$$

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. 2.12:

$$U \otimes \mathbb{1} |00\rangle = \begin{pmatrix} a & 0 & b & 0 \\ 0 & a & 0 & b \\ c & 0 & d & 0 \\ 0 & c & 0 & d \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} a \\ 0 \\ c \\ 0 \end{pmatrix} \quad (2.14)$$

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

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

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. 2.15 which can be reexpressed as follows,

$$\begin{pmatrix} a \\ 0 \\ c \\ 0 \end{pmatrix} = a \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 0 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + c \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + 0 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = a|00\rangle + 0|01\rangle + c|10\rangle + 0|11\rangle \quad (2.16)$$

This can be factorized into the following tensor product:

$$a|00\rangle + c|10\rangle = (a|0\rangle + c|1\rangle) \otimes |0\rangle \quad (2.17)$$

In contrast, consider the following state:

$$|\phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad (2.18)$$

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. 2.18 and knowing that measurement collapses a superposition the post-measurement (PM) state $|\phi\rangle_{PM}$ is given by:

$$|\phi\rangle_{PM} = |11\rangle \quad (2.19)$$

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:

$$\sigma_1 = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (2.20)$$

The action of a quantum gate on the arbitrary qubit state $|\psi\rangle$ (Equ. 2.3) can be analysed using the gate's matrix and the qubit's vector representation. Applying some straightforward linear algebra yields:

$$X \otimes |\psi\rangle = X \otimes (\alpha|0\rangle + \beta|1\rangle) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} = \beta|0\rangle + \alpha|1\rangle \quad (2.21)$$

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,

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

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,

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

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

From Equ. 2.21, 2.22 and 2.23 it follows that X is its own inverse as well as its own Hermitian conjugate:

$$XX = XX^\dagger = \mathbb{1} \quad (2.24)$$

$$X = X^{-1} = X^\dagger \quad (2.25)$$

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.

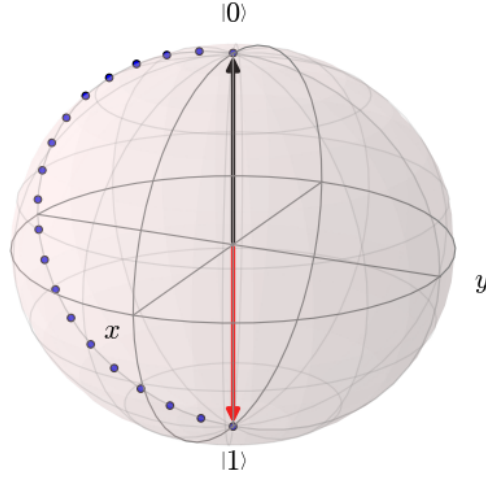


Figure 2.3: Visualization of the X gate on the Bloch sphere. Initial state (black) transformed to final state (red).

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

Gate	Name	Circuit representation	Matrix	Description	Rotation	Bloch sphere
$\mathbb{1}$	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}	

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:

$$CNOT = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.26)$$

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,

$$|\phi\rangle = |0\rangle \otimes |0\rangle = |00\rangle \quad (2.27)$$

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

$$(H \otimes \mathbb{1})|\phi\rangle = (H \otimes \mathbb{1})|00\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle \quad (2.28)$$

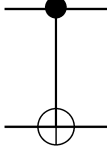
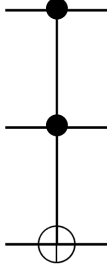
Now consider applying the CNOT to this state whereby the control qubit is coloured **red** and the target qubit **green**.

$$CNOT(\text{red}, \text{green}) \otimes \left(\frac{1}{\sqrt{2}}|\text{red}0\rangle + \frac{1}{\sqrt{2}}|\text{red}1\rangle \right) = \frac{1}{\sqrt{2}}|\text{red}0\rangle + \frac{1}{\sqrt{2}}(\mathbb{1} \otimes X)|\text{red}1\rangle = \frac{1}{\sqrt{2}}|\text{red}0\rangle + \frac{1}{\sqrt{2}}|\text{red}1\rangle \quad (2.29)$$

The last expression in Equ. 2.29 was used as an example (Equ. 2.18) 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 described in (Schuld, Sinayskiy, & 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 and 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.



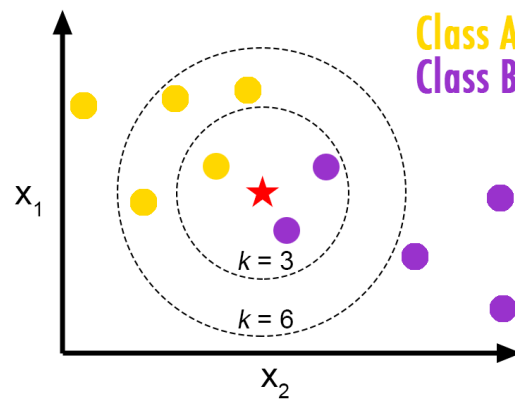
Figure 2.4: Pictures of apple and banana from training data set.³

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 features might be the RGB value of the most frequent pixel colour since bananas and apples have different colour spectra or a measure of the curvature of the main object in the picture since an apple is almost spherical whereas a banana looks more like a bend line. Through the selection & extraction of features the dimension 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* v_j .

Mathematically, the training data set D_T consists of ten feature vectors v_0, v_1, \dots, 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.

The kNN algorithm is a non-parametric classifier that given a new unclassified input vector x (red star in Fig. 2.5) considers the k nearest neighbours within the training set (using a predefined measure of distance) and classifies 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 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 $k = 10$, 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.

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.

Chapter 3

Methods

All the calculations and plots for this thesis are done in the programming language Python which has a very intuitive syntax and comes with large libraries for scientific computing and plotting.

For the implementation of the quantum kNN algorithm there are two fundamentally different ways: Running it 1) by simulating a QC or 2) 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 (verify and cite!). It comes with a large palette of predefined single and multi 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.

```

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|>

3.2 IBM Quantum Experience

IBM QASM

Earlier this year IBM has enabled public cloud access to their experimental quantum processor containing five non error-corrected superconducting qubits located at the Watson Research Center in New York (VERIFY AND CITATION). 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. ?? consists of 5 horizontal lines, one for each qubit, and enables the user to choose from a universal gate set consisting of 10 quantum logic gates (VERIFY!). 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.

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

The main limitation of the IBM Quantum Experience are the qubit decoherence times since they restrict the maximum number of possible operations before the qubits lose their quantum behaviour and their quantum information. Thus, the number of quantum gates is currently limited to only 40 which essentially means 39 logic gates and 1 measurement gate. The amplitude damping times of the five qubits range from 49.5 μ s to 85.3 μ s. Furthermore, the phase damping times range from 56.0 μ s to 139.7 μ s. The implementation of a single qubit quantum logic gate takes 130ns and applying a CNOT gate takes 500ns. CITATIONS FOR ALL THESE NUMBERS!

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 Algorithms

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 vector v :

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

Encoding classical data into qubits

A k -dimensional probability vector requires $4k$ classical bits which are encoded one-to-one into $4k$ qubits, e.g.

Multiply vector by ten such that probabilities can easily be represented in binary,

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

Convert each entry to binary,

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

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

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

Apply X gate to respective qubits:

$$n = 01100100 \rightarrow |n\rangle = |01100100\rangle \quad (4.5)$$

Only slight speed up possible
Number of qubits increases linearly with size of classical data

Encoding classical data into amplitudes

k -dimensional probability vector is encoded into only $\log_2(k)$ qubits since the number of amplitudes grows exponentially with the number of qubits e.g.

$$\begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \rightarrow |n\rangle = \sqrt{0.6}|0\rangle + \sqrt{0.4}|1\rangle \quad (4.6)$$

Possibility of exponential speed up
Number of qubits only grows logarithmically with size of classical data
Non-trivial algorithms needed to initialize amplitude distribution

Grover & Rudolph

Diffusion matrix from quantum random walks

Trugenberger et al.

Schack paper Maria mentioned but which I haven't used and looked at.

4.0.2 Quantum k-nearest Neighbour Algorithm

$$\frac{1}{\sqrt{2M}} \sum_{m=1}^M (|0\rangle |\Psi_{\tilde{x}}(\star)\rangle + |1\rangle |\Psi_{x^m}\rangle) |y^m(A \text{ or } B)\rangle |m\rangle \quad (4.7)$$

where

$$|\Psi_{\tilde{x}}(\star)\rangle = \sum_{i=1}^N \tilde{x}_i |i\rangle \quad |\Psi_{x^m}\rangle = \sum_{i=1}^N x_i^m |i\rangle \quad (4.8)$$

$$e.g. \quad \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \rightarrow |n\rangle = \sqrt{0.6} |0\rangle + \sqrt{0.4} |1\rangle \quad (4.9)$$

Applying the **Hadamard gate** interferes the input and the training vectors:

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^M (|0\rangle [|\Psi_{\tilde{x}}\rangle + |\Psi_{x^m}\rangle] + |1\rangle [|\Psi_{\tilde{x}}\rangle - |\Psi_{x^m}\rangle]) |y^m(A \text{ or } B)\rangle |m\rangle \quad (4.10)$$

→ Perform **conditional measurement** on ancilla qubit.

Successful if $|0\rangle$ state is measured.

After successful conditional measurement, the state is proportional to

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^M \sum_{i=1}^N (\tilde{x}_i + x_i^m) |0\rangle |i\rangle |y^m(A \text{ or } B)\rangle |m\rangle \quad (4.11)$$

Probability to measure class B:

$$p(|y^m\rangle = |1(B)\rangle) = \sum_{m|y^m=1(B)} 1 - \frac{1}{4M} |\tilde{x} - x^m|^2 \quad (4.12)$$

Hereby, the advantages of the quantum version are the parallel computation of the distances between each training vector and the input vector as well as contracting distance computation and distance weighting into one computational step.

Chapter 5

Results and Discussion

Controlled U Gate

Often there is a need for applying certain quantum gates in a controlled manner. Thus a controlled U (CU) gate is required whereby U can be any unitary single-qubit gate. The CU gate is defined as:

$$CU = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & U \end{pmatrix} \quad (5.1)$$

It is important to note that the CNOT gate is essentially a CU gate in the case of $U = X$.

Most of the time the CU gate cannot be implemented directly and has to be realized through larger quantum circuits consisting of CNOT and single-qubit gates. (1, 2) describe such a decomposition as shown in Fig. 5.1.

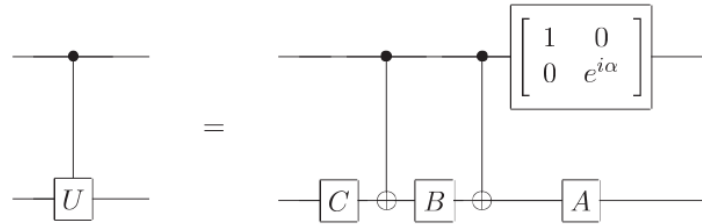


Figure 5.1: Circuit decomposition for a controlled-U operation for single-qubit gate U.³

The idea is that when the control qubit is $|0\rangle$ the gate combination ABC is applied to the target qubit and has to equal the identity gate:

$$ABC = \mathbb{1} \quad (5.2)$$

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,

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

$$e^{i\alpha}AXBXC = U \tag{5.3}$$

In order to find the matrices A,B,C and the additional parameter α the following equation has to be solved:

$$U = \begin{pmatrix} e^{i(\alpha - \frac{\beta}{2} - \frac{\delta}{2})} \cos \frac{\gamma}{2} & -e^{i(\alpha - \frac{\beta}{2} + \frac{\delta}{2})} \sin \frac{\gamma}{2} \\ e^{i(\alpha + \frac{\beta}{2} - \frac{\delta}{2})} \sin \frac{\gamma}{2} & e^{i(\alpha + \frac{\beta}{2} + \frac{\delta}{2})} \cos \frac{\gamma}{2} \end{pmatrix} \tag{5.4}$$

Chapter 6

Outlook

Chapter 7

Conclusion

Chapter 8

Personal reflection

References

- Bekkerman, R., Bilenko, M., & Langford, J. (2011). *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press.
- 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
- 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.
- IBM. (2016). *What is big data?* <https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>. (Accessed: 2016-09-08)
- 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
- 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
- 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., 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). Polynomial-time algorithm for prime factorization and discrete logarithms on a quantum computer: Proc. , 124.