Quantum machine learning for supervised pattern recognition

How quantum computers learn from data

by

MARIA SCHULD

Submitted in fulfilment of the academic requirements of the Doctor of Philosophy in the School of Chemistry and Physics, University of KwaZulu-Natal.

Supervised by Prof. Francesco Petruccione and Dr. Ilya Sinayskiy.

Quantum Research Group
School of Chemistry & Physics
College of Agriculture, Engineering and Science
University of KwaZulu-Natal
Durban, South Africa

August 21, 2017

Abstract

Humans are experts at recognising patterns in past experience and applying them to new tasks. For example, after seeing pictures of a face we can usually tell if another image contains the same person or not. Machine learning is a research discipline at the intersection of computer science, statistics and mathematics that investigates how pattern recognition can be performed by machines and for large amounts of data. Since a few years machine learning has come into the focus of quantum computing in which information processing based on the laws of quantum theory is explored. Although large scale quantum computers are still in the first stages of development, their theoretical description is well-understood and can be used to formulate 'quantum software' or 'quantum algorithms' for pattern recognition. Researchers can therefore analyse the impact quantum computers may have on intelligent data mining. This approach is part of the emerging research discipline of *quantum machine learning* that harvests synergies between quantum computing and machine learning.

The research objective of this thesis is to understand how we can solve a slightly more specific problem called *supervised* pattern recognition based on the language that has been developed for universal quantum computers. The contribution it makes is twofold: First, it presents a methodology that understands quantum machine learning as the combination of data encoding into quantum systems and quantum optimisation. Second, it proposes several quantum algorithms for supervised pattern recognition. These include algorithms for convex and non-convex optimisation, implementations of distance-based methods through quantum interference, and the preparation of quantum states from which solutions can be derived via sampling. Amongst the machine learning methods considered are least-squares linear regression, gradient descent and Newton's method, k-nearest neighbour, neural networks as well as ensemble methods. Together with the growing body of literature, this thesis demonstrates that quantum computing offers a number of interesting tools for machine learning applications, and has the potential to create new models of how to learn from data.

PREFACE

The research contained in this dissertation was completed by the candidate while based in the Discipline of Physics, School of Chemistry & Physics of the College of Agriculture, Engineering and Science, University of KwaZulu-Natal, Westville Campus, Durban, South Africa.

The research was financially supported by the European Commission, the German Academic Foundation as well as the University of KwaZulu-Natal and the National Research Foundation of South Africa.

The contents of this work have not been submitted in any form to another university and, except where the work of others is acknowledged in the text, and the results reported are due to investigations by the candidate.

Signed:

Date:

DECLARATION 1: PLAGIARISM

I, Maria Schuld, declare that:

(i) the research reported in this dissertation, except where otherwise indicated or acknowledged, is my original work;

(ii) this dissertation has not been submitted in full or in part for any degree or examination to any other university;

(iii) this dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons;

(iv) this dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

1. their words have been re-written but the general information attributed to them has been referenced;
2. where their exact words have been used, their writing has been placed inside quotation marks, and referenced;

(v) where I have used material for which publications followed, I have indicated in detail my role in the work;

(vi) this dissertation is primarily a collection of material, prepared by myself, published as journal articles or presented as a poster and oral presentations at conferences. In some cases, additional material has been included;

(vii) this dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

Signed:

Date:

DECLARATION 2: PUBLICATIONS AND PRESENTATIONS

The doctoral project has led to the following publications so far. Unless indicated below, my role in each paper was to conduct the underlying research and to produce the publication. Where content of a publication was used in a chapter of this thesis, the reference is pointed out at the beginning of the chapter.

Publications

- Maria Schuld, Francesco Petruccione (2017) Quantum ensembles of quantum classifiers. *In preparation.*
- Maria Schuld, Mark Fingerhuth, Francesco Petruccione (2017) Quantum machine learning with small-scale devices: Implementing a distance-based classifier with a quantum interference circuit.[1] ArXiv preprint quant-ph/1703.10793. *Submitted to Physical Review Letters.*
- Patrick Rebentrost, Maria Schuld, Francesco Petruccione, Seth Lloyd (2016) Quantum gradient descent and Newton's method for constrained polynomial optimization.[2] ArXiv preprint quant-ph/1612.01789. *About to be submitted.*
- Maria Schuld (2017) A quantum boost for machine learning. IOP Physics World, March issue, pp. 28-31. *Invited article.*
- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2016) Prediction by linear regression on a quantum computer, Physical Review A, 94, 2, 022342.
- Maria Schuld, Francesco Petruccione (2016) Quantum machine learning, in: Sammut, Claude, and Geoffrey I. Webb, eds. Encyclopaedia of Machine Learning and Data Mining. Springer Science & Business Media. *Invited encyclopaedia entry.*
- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2015) How to simulate a perceptron using quantum circuits, Physics Letters A, 379, pp. 660-663.
- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2015) Introduction to quantum machine learning, Contemporary Physics, 56, 2, pp. 172-185.
- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2014) Quantum computing for pattern classification. Pham, Duc-Nghia, Park, Seong-Bae (Eds.) Lecture Notes in Computer Science Vol 8862, Springer International Publishing, pp. 208-220.
- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2014) Quantum neural networks - Our brain as a quantum computer?, in: Chris Engelbrecht and Steven Karataglidis (eds.), Proceedings of SAIP2014, the 59th annual conference of the South African Institute of Physics, 7-11 July 2014 Johannesburg.
- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2014) The quest for a quantum neural network, Quantum Information Processing 13, 11, pp. 2567-2586.

[1] In collaboration with Mark Fingerhuth who contributed the design and implementation of the quantum circuit on the IBM Quantum Experience.

[2] Patrick Rebentrost and myself were equally involved in research and write up. Seth Lloyd contributed towards the idea and manuscript.

- Maria Schuld, Ilya Sinayskiy, Francesco Petruccione (2014) Quantum walks on graphs representing the firing patterns of a Quantum Neural Network, Physical Review A, 89, 032333. *The material is partly based on my Master's thesis.*

Presentations, conferences and research visits

- American Physical Society March Meeting, New Orleans, United States, 12 - 17 March (Poster: Convex and non-convex optimisation in quantum machine learning.)
- Research visit to Prof Hans Briegel at the University of Innsbruck, Austria, 7 March 2017 (Talk: Quantum Machine Learning: An interference circuit for classification.)
- Quantum Machine Learning Summer School, Drakensberg, South Africa, 23 January - 1 February 2016 (Lectures: Introduction to quantum machine learning/ Linear algebra with quantum systems.)
- UKZN College of Agriculture, Engineering and Science Research Day 29 November 2016 (Talk: Quantum machine learning - how to learn from data using quantum computers.)
- Quantum Machine Learning Conference at the Perimeter Institute, Waterloo, Canada, 8-13 August 2016 (Invited Talk: Prediction on a quantum computer: Linear Regression.)
- Quantum Machine Learning Conference, North Coast, South Africa, 18-22 July 2016 (Talk: Prediction on a quantum computer: Ensemble Methods.)
- Research visit to Prof Bernhard Schölkopf at the Max-Planck-Institute for Intelligent Systems in Tübingen, Germany, 30 June -3 July 2016
- Research visit to Prof Seth Lloyd at the Massachusetts Institute of Technology, Boston, USA, 13-21 December 2015
- Quantum Machine Learning Workshop at the NIPS 2015 conference, Montreal, Canada, 12 December 2015
- College of Agriculture, Engineering and Science Research day, 22 October 2015 (Talk: Towards a quantum algorithm for regression.)
- Summer School of the German Academic Foundation 2-15 August 2015, Working Group 'Causality or how to Learn from Data' in Neubeuern, Germany (Talk: Counterfactuals.)
- Conference Quantum Effects in Biological Systems, Florence, Italy, 29 June - 2 July 2015 (Poster: Quantum Extreme Learning Machines.)
- Research visit to Dr Luca Turin at the University of Ulm, Germany, 5 March 2015 (Talk: Quantum Machine Learning or how to use qubits to recognise a cat.)
- Research visit to Prof Uwe Jaekel at the University of Koblenz, RheinAhrCampus Remagen, Germany 3-4 March 2015 (Talk: Quantum Machine Learning or how to use qubits to recognise a cat.)
- Workshop on Quantum Computing for Artificial Intelligence at the Pacific Rim International Conference on Artificial Intelligence, Goldcoast, Australia, 1-5 December 2014 (Tutorial: Quantum Machine Learning/ Talk: Quantum computing for pattern classification.)

- Pacific Rim International Conference on Artificial Intelligence, Goldcoast, Australia, 1-5 December 2014 (Talk: Quantum computing for pattern classification.)
- Quantum Simulations and Quantum Walks Conference, South Coast, South Africa, 24-28 November 2014 (Talk: Using quantum walks to model a quantum associative memory.)
- Quantum Information, Communication and Control Conference, Drakensberg, South Africa, 3-7 Nov 2014 (Talk: How to implement an artificial neural network on a quantum computer.)
- UKZN School of AES Research day 27 October 2014 (Talk: Quantum machine learning and its application in data mining.)
- Quantum Africa 3 Conference, Rabat, Morocco, 22-26 September 2014 (Talk: Perceptrons on a quantum computer.)
- UKZN Quantum Research Group 9 & 10 June 2014 (Lecture series: Quantum machine learning.)
- SAIP Conference, University of Johannesburg, 7-11 July 2014 (Talk: Quantum neural networks - Our brain as a quantum computer?)
- Coherent Control of Complex Quantum Systems C3QS Workshop, Okinawa, Japan, 14-17 April 2014 (Poster: Quantum Neural Networks - Using quantum walks to build a quantum associative memory.)

Signed:

Date:

Acknowledgements

I want to thank Francesco Petruccione and Ilya Sinayskiy, not only for being my dedicated supervisors but also my treasured mentors over the last few years. It was your constant support that allowed me to grow as a scientist and that taught me many tricks of the trade. I thank my mother, father, brother, sister and grandmother, as well as Franzi, Sarah, Carina and Alex for allowing me to live so far away from them. Thanks to Jane for being my family in South Africa. Thanks to the wonderful people from UKZN's physics department, especially to Betony, Judy, Sharmini, Mark, Samke, Nonky, Louise, Hazma and Samah for the good times we shared; and thanks to Diane and Neli for their support. Thanks to Nori, Shelby, and Santiago for sharing a home.

Lastly, Chris, thanks for being there at my side.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning, on the one hand, is the art and science of making computers learn from data how to solve problems instead of being explicitly programmed. *Quantum computing*, on the other hand, describes information processing with devices based on the laws of quantum theory. Both machine learning and quantum computing are expected to play a substantial role in how society deals with information in the near future. It is therefore only natural to ask how they could be combined, a question explored in the emerging discipline of *quantum machine learning*. One approach to quantum machine learning is to use quantum computers for machine learning applications. Although still limited to small-scale setups in the research laboratories, we already know the language quantum computers speak and can formulate the 'software' they may run. In other words, we can develop quantum algorithms for machine learning tasks which allow us to investigate how quantum computers can learn from data. This thesis is a theoretical investigation of the potential and the challenges of quantum machine learning and develops examples of quantum machine learning algorithms that, together with other recent results, give some first evidence that there is more to expect from data mining with quantum information.

1.1 Background

Computers are physical devices based on electronic circuits that process information. Algorithms (the computer programs or 'software') are on a basic level not more than recipes of how to manipulate the current in these circuits on the hardware level in order to execute computations, and we can describe these processes with a classical theory of physics. In principle, microscopic systems such as photons, electrons and atoms can also be used to build computers and define algorithms for computations. But they require another mathematical description to capture the fact that on small scales, nature behaves radically different from what our intuition teaches us. This mathematical framework is called *quantum theory* and since its development at the beginning of the 20th century it is generally considered to be the most comprehensive description of microscopic physics that we know of. A computer that can only be described with the laws of quantum theory is called a *quantum computer*.

Since the 1990s quantum physicists and computer scientists have been analysing how quantum computers can be built and what they could potentially be used for. Building a quantum computer in the laboratory is not an easy task, as it requires to control very small systems accurately, but

without disturbing the highly fragile objects and hence destroy the quantum effects that we want to harvest. A lot of exciting progress is made in the development of small-scale devices, and, many experts believe that large-scale quantum computing is only a matter of time. The research field has since long left a purely academic interest inside the quantum physics community and is co-developed in the research labs of large IT companies such as *Google*, *Microsoft* and *IBM*.[1] While technological implementations are advancing, quantum physicists have developed languages to describe computations executed by a quantum system on a fundamental level, languages that allow us to investigate these devices from a theoretical perspective. An entire 'zoo'[2] of *quantum algorithms* has been proposed and is waiting to be used on physical hardware. While some of these quantum algorithms promise to be more efficient for certain computations than classical algorithms, the challenge lies in finding real-life applications that can exploit these advantages.

Machine learning is another discipline causing a lot of excitement in the academic world as much as in the IT sector. In this combination of statistics, mathematics and computer science, computers have to learn from prior examples - usually large datasets based on highly complex and nonlinear relationships - how to make predictions or solve unseen problem instances. Machine learning was initially born as the 'data-driven' side of artificial intelligence research and tried to give machines human-like abilities such as image recognition, natural language processing and decision making. While such tasks come natural to humans, we do not know in general how to make machines obtain similar skills. For example, looking at an image of a cat it is unclear how to relate the information that the pixel (543,1352) is dark red to the concept of a cat. Machine learning approaches this problem by making the computer recover patterns from data and use these patterns to solve a new problem.

As data is becoming increasingly accessible, machine learning systems become better and are integrated in many spheres of our day-to-day life. For example, they scan through huge amounts of emails every day in order to pick out spam mail, or through masses of images on social platforms to identify offensive contents. They are used in forecasting of macroeconomic variables, risk analysis as well as fraud detection in financial institutions, and medical diagnosis.[3] What is celebrated as new 'fundamental breakthroughs' is thereby often based on the growing sizes of datasets as well as computational power, and not due to genuine innovation[4]. Methods such as neural networks, support vector machines or Adaboost, as well as the latest trend towards deep learning have basically been invented in the 1990s and earlier. Finding new approaches is difficult as many tasks translate into very difficult optimisation problems. To solve them, computers have to search more or less blindly through a vast landscape of solutions to find the best candidate. A lot of research therefore focuses on finding variations and approximations of methods that work well in practise, and machine learning is known to contain a fair share of 'black art' [1].

In recent years, there is a growing body of literature with the objective of combining the two

[1] `http://www.technologyreview.com/featuredstory/531606/microsofts-quantum-mechanics/`, last visited December 2015

[2] http://math.nist.gov/quantum/zoo/, last visited February 2017.

[3] The growing social impact of machine learning becomes apparent when footage from surveillance cameras can one day be analysed fast enough to track the whereabouts of any person at any time around the globe, or when job markets are restructured by replacing low skill labour with machines. These are controversial issues that we urgently have to start debating, but they are not discussed in this thesis.

[4] Private conversations, for example with Hans-Georg Zimmermann from Siemens, Prof Bernhard Schólkopf from the Max-Planck-Institute for Intelligent Systems.

disciplines of quantum information processing and machine learning. Quantum machine learning has initially been used to refer to questions of learning with quantum computing devices or agents. A broader definition of the field includes proposals that think vice versa and use well-established tools of machine learning to control and learn about quantum systems. When not otherwise stated, this thesis will follow the initial, narrow definition, sometimes referred to as *quantum-assisted* or *quantum-enhanced machine learning* [2].

Investigations into quantum machine learning were only sporadic and scattered[5] before 2014 and at the beginning of this PhD research project, when their number suddenly started growing. Only three years later, various international workshops and conferences[6] have been organised on the topic, and a range of groups in quantum information science started research projects and collaborations. Combining a dynamic multi-billion dollar industry with the still 'mysterious' and potentially profitable technology of quantum computing has also sparked a lot of interest in industry. Illustrative examples are Google's *Quantum Artificial Intelligence Lab*[7] established in 2013, the *Quantum Optimisation and Machine Learning*[8] project of Oxford University in cooperation with *Nokia* and *Lockheed Martin* launched in 2014, as well as active research in Microsoft's *Quantum Architectures and Computation*[9] group. As a consequence, a variety of results have been established and suggest that quantum machine learning is a promising research discipline in the making.

1.2 Problem definition and research objective

The central problem addressed in this thesis is the analysis of machine learning from the perspective of quantum computing as well as the development of new quantum machine learning algorithms. A number of crucial delineations limit the scope of this study and shall be outlined briefly here.

As mentioned above, there are various ways one can combine machine learning and quantum computing. A useful typology has been introduced by Aimeur, Brassard and Gambs [3] and has been adapted by Dunjko, Taylor and Briegel [4] in a slightly different context. It distinguishes four cases depending on whether the data and the information processing device are each quantum (Q) or classical (C). The case CC refers to classical data being processed classically (i.e., conventional machine learning), but possibly with methods borrowed from quantum information. A central example is the use of tensor networks developed for quantum many-body-systems in the context of neural network training [5]. The case QQ looks at 'quantum data' (i.e. the outputs of an experiment on a quantum system) being processed by a quantum computer. This might require a reformulation of the pattern recognition problem for quantum systems [6], and is the least explored of the four approaches.[10] The case QC investigates how machine learning can help with quantum

[5]The only exception were attempts to find quantum models for neural networks, which had a more consistent history dating back to the 1990s and being largely inspired by biological considerations.

[6]Such as a workshop at the *Neural Information Processing Systems (NIPS)* conference in Montreal, Canada in December 2015, our own Quantum Machine Learning Workshop in South Africa in July 2016 as well as a Quantum Machine Learning conference at the Perimeter Institute in Waterloo, Canada, in August 2016.

[7]`https://plus.google.com/+QuantumAILab/posts`, [last visited January 2017]

[8]`https://quopal.com/`, [last visited January 2017]

[9]`http://research.microsoft.com/en-us/groups/quarc/default.aspx`, [last visited January 2017]

[10]In a reinforcement learning setting, the case of a quantum agent acting in a quantum environment has been discussed by [2].

information processing such as state tomography, state discrimination or learning about phases of matter, and also shows a growing body of literature [7, 8, 9, 10]. This thesis focuses exclusively on the last case, CQ, which is a classical machine learning task based on a 'conventional' dataset, and the task has to be solved using a quantum computer (possibly together with classical computation).

As a second limitation to the scope, I largely focus on a specific machine learning task called *supervised pattern recognition*. In supervised learning, a mapping between inputs and outputs has to be inferred from example data of input-output pairs ('labeled data'). This covers a large share of the machine learning literature and can be regarded as the problem which is best understood. It ignores other interesting areas such as *unsupervised learning* using no outputs to guide the training, or *reinforcement learning* based on agents, strategies and reward systems.

The term 'quantum computing' can in general refer to any kind of physical system governed by quantum theory whose dynamics are used to solve a computational problem. However, here I will mostly rely on the universal *circuit* or *gate model* known from many introductory textbooks into quantum computation. Second, the goal of quantum machine learning is usually to show 'advantages' compared to purely classical methods. This term could have several meanings. For example, one can improve machine learning by making models more likely to be correct, more robust against errors, requiring fewer samples from which to learn, or being executed on a computer in less time. For the development of algorithms I will mostly focus on improvements in terms of computational complexity and runtime, that is, I will interpret 'better' as *asymptotically faster*[11].

The central research question can now be stated as follows:

Research question: *How can we solve supervised pattern recognition tasks with gate-based universal quantum computing, and what advantages for the scaling of the asymptotic runtime can we get compared to machine learning with classical computers?*

The thesis will argue throughout that machine learning poses two challenges to quantum information. The first challenge is that of encoding datasets into quantum states. The typical structure shared by all machine learning algorithms is that they extract "small" amounts of information (in the extreme, a one bit yes-no decision to predict a new instance) from large inputs, which can for the famous cases of *big data* mining refer to billions of input values. Quantum machine learning consequently has to establish theoretical frameworks, software and hardware for state preparation and control that feed the data into the quantum system in order to process it. Information encoding will also be an important category to distinguish different types of quantum machine learning algorithms.

The second challenge lies in solving the optimisation problems that are the heart of most supervised pattern recognition tasks. These optimisation problems can have various mathematical solution methods such as matrix inversion, computing gradients or search, and different tools from quantum information processing can be applied. Note that while so far most quantum machine learning proposals (including those in this thesis) try to use quantum tools to solve the optimisation problems

[11] Asymptotic complexity describes how computational resources of time and space grow when we make the problem bigger (i.e. consider more data).

derived for classical models and tailor-made for the needs of classical computers, one of my central conclusions is that we have to start thinking about models and optimisation problems that suit the quantum tools and devices right from the beginning. In other words, instead of solving a problem designed for a classical computer, we should design quantum models that fit quantum architectures. This will be an important paradigm change for future generations of quantum machine learning algorithms.

1.3 Structure of the dissertation

With these two challenges in mind I will use two different ways to investigate the research question. First, I will systematically review relevant contributions produced by the quantum machine learning community up to today, aggregating the findings and working out how different choices to tackle these two challenges give rise to four different approaches to quantum machine learning. The second stage forms the main contribution of the thesis and develops a number of quantum algorithms for supervised pattern recognition as demonstrations of how we can do machine learning on quantum computers, and analyses their potential computational advantages. I expect the reader to have a firm background in quantum theory but not necessarily in machine learning, which is mirrored in how the theoretical foundations are presented.

To structure the material, the thesis is divided into three parts:

- Part I introduces the theoretical foundations, where
  - Chapter 2 presents concepts and methods needed from machine learning
  - Chapter 3 present concepts and methods needed from quantum computing
- Part II contains methodological foundations of quantum machine learning as well as a literature review.
  - Chapter 4 addresses the problem of data encoding into quantum systems and summarises important approaches that the literature frequently makes use of
  - Chapter 5 defines the term 'quantum advantages' further, and investigates three different ways in which quantum information can lead to an advantage with reference to related literature
  - Chapter 6 presents the main literature review and identifies four different approaches to quantum machine learning algorithmic design
- Part III contains the main original contributions that have mostly been published in peer-reviewed journals
  - Chapter 7 presents a quantum algorithm for linear regression with the central problem of finding the singular value decomposition of a data matrix
  - Chapter 8 explores optimisation with two iterative methods, namely gradient descent or Newton's method, on a quantum computer
  - Chapter 9 demonstrates two quantum versions of k-nearest neighbour in which distances between data points are written into the amplitudes of a quantum state and evaluated for prediction

- Chapter 10 gives a framework to think about the construction of neural networks, and introduces a way to execute a nonlinear activation function with well-known quantum tools
- Chapter 11 contains the description and outlook to a strategy of using quantum superposition for the parallel evaluation of models that appears in ensemble methods

Chapter 12 will present a synopsis of the preceding chapters, draw a conclusion and show some possible avenues forward.

Note that many contributions to the literature are not or not yet published in peer-reviewed journals but accessible through the ArXiv e-print archive hosted by Cornell University. Especially researchers at IT companies sometimes make their results public in this format instead of submitting their work to a journal. Still, many of these articles are important sources and will therefore be included in the bibliography. Also note that I will continue to make use of the first person singular despite of what is taught in some style guidelines for dissertation writing that prefer to keep up the illusion of objectivity.

Part I

Foundations

Chapter 2

Machine learning

Machine learning originally emerged as a sub-discipline of artificial intelligence research where it extended other areas such as perception, communication and reasoning. For humans, learning means to find patterns in previous experience which help us to deal with an unknown situation. For example, someone who lived on a farm for 30 years will be very good at predicting the weather for the rest of the day. Financial experts pride themselves of being able to predict the immediate stock market trajectory. When speaking about machines, previous experience can be translated to 'data' (for example information on wind, perspiration and temperature, or macroeconomic variables), while the solution to a new problem may be understood as the output of an algorithm.

Although understanding and reproducing human-like behaviour is still frequently referred to by the machine learning community[1], latest in the 1990's machine learning had outgrown its foundations and became an independent discipline that has -with some intermediate recessions- been expanding ever since. Today, machine learning is another word for data-driven decision making or prediction. Like in the weather and stock market examples, the patterns from which the predictions have to be derived are usually very complex and we ourselves have only little understanding of how our intuition works. The challenge in machine learning is to teach computers how to generically find and use patterns in data, even though we do not know them ourselves. This is an important distinction to the closely related discipline of statistics, where models are used to test hypotheses and understand the rules of the underlying system better (for example the relationship between smoking and lung cancer). Still, machine learning heavily draws on methods developed in statistics. Other important parent disciplines are computer science and mathematics with an interesting interplay between the two: Computer algorithms are used to solve mathematical problems which therefore need to be computable. At the same time, mathematics is used to describe what algorithms do and algorithmic thinking can lead to new mathematical models.

This chapter is an attempt to give a quantum physicist access to the major concepts in the vast landscape of machine learning research. An excellent textbook for further reading is written by Christopher Bishop [12], but many others were also used as a basis for this chapter [13, 14, 15, 16, 17, 18, 19]. The first Section 2.1.1 introduces to the central task considered in this thesis and gives

[1] As the recently established connections between deep neural networks and the visual cortex of our brain demonstrate [11].

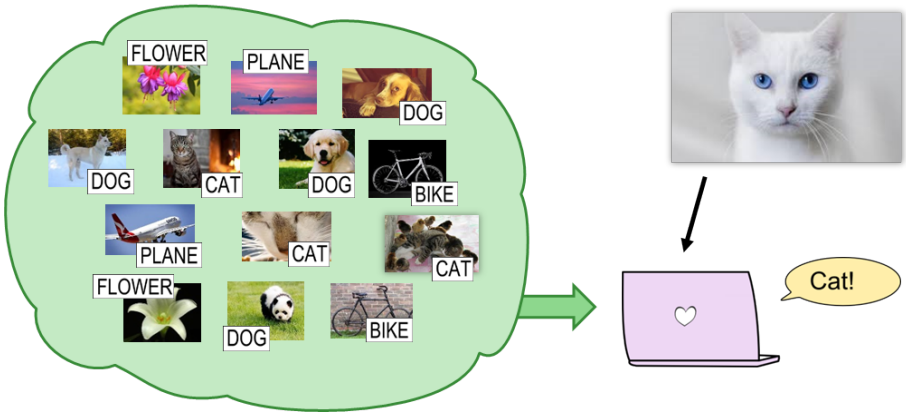


Figure 2.1: Illustration of Example 2.1.1. While for humans, recognising the content of pictures is a natural task, for computers it is much more difficult to make sense of their numerical representation. Machine learning feeds example images and their content label to the computer, which learns the underlying structure to classify previously unseen images.

an idea of how models are derived from data. The second Section 2.2 goes into more detail and distinguishes between four different types of models that define four slightly different approaches to training. The last section goes into more detail for a number of models from these four types. These models will become important when discussing quantum machine learning algorithms in later chapters, and cover a substantial share of the spectrum found in most standard textbooks. Readers familiar with machine learning can skip this chapter and read selected sections only when it becomes necessary.

2.1 A gentle introduction to prediction

Almost all machine learning algorithms have one thing in common: They are fed with data and produce the answer to a question. In extreme cases the datasets can consist of billions of values while the answer is only one single bit. The term 'answer' can stand for many different types of outputs. For example, when future values of a time series have to be predicted, it is a *forecast*, and when the content of images is recognised it is a *classification*. In the context of medical or fault finding applications the computer produces a *diagnosis* and if a robot has to act in an environment one can speak of a *decision*. Since the result of the algorithm is always connected to some uncertainty, another common expression is a *guess*. I refer to all these terms as the *output*, *label* or *prediction* of the model, while the data is considered to be the *input*.

2.1.1 Four examples for prediction tasks

Four introductory examples of typical prediction problems shall serve as a motivation for the wide scope of machine learning.

Example 2.1.1: Image recognition

The introduction mentioned the task of recognising a visual input such as an image. While our brain seems to be optimised to recognise concepts such as 'house' or 'mountain panorama' from optical stimuli, it is not obvious how to program a computer to do the same, as the relation between the pixels' RGB values and the image's content can be very complex. In machine learning one does not try to explicitly implement such an algorithm, but presents a large number of already labeled

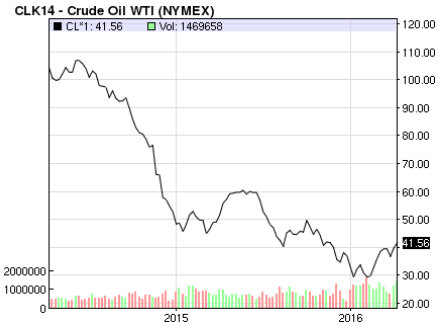


Figure 2.2: Given the graph of the global oil price in the last two years (and probably also the values for other correlated macroeconomic variables), time series forecasting has to predict the future development to take a decision when it is best to buy or sell oil. Source: www.nasdaq.com, 14.4.2016.

images to the computer from which it is supposed to learn the relationship of the digital image representation and its content (see Figure 2.1). In other words, the complex and unknown input-output function of *pixelmatrix* $\rightarrow$ *content of image* has to be approximated. The 'fruit-fly example' for image recognition is the well-known MNIST dataset consisting of black and white images of handwritten digits that have to be identified automatically. Current algorithms guess the correct digit with a success rate of up to 99.65%.[2] An important real-life application for handwritten digit recognition is the processing of postal addresses on mail.

Example 2.1.2: Time series forecasting

A time series is a set of data points recorded in consecutive time intervals. An example is the development of the global oil price (Figure 2.2). Imagine that for every day in the last two years one also records the values of important macroeconomic variables such as the gold price, the DAX index and the Gross Domestic Products of selected nations. These indicators will likely be correlated to the oil price, and there will be many more independent variables that are not recorded. In addition, the past oil price might itself have explanatory power of any consecutive one. The task is to predict on which day in the upcoming month oil will be cheapest. This is an important question for big companies who use large amounts of natural resources in their production, and Siemens is but one example where the recommendation of neural networks are considered for large-scale purchases.

Example 2.1.3: Medical diagnosis

When diagnosing a patient, doctors frequently use data measured by Magnetic Resonance Imaging (MRI), X-Ray or Electrocardiography (ECG) devices. A combined human-machine learning approach would be to run software on this data to produce a computer generated value for the chance of the patient suffering from a given disease. Such machine-aided medical applications are a growing market, and can support the doctor's intuition in these important decisions. The machine learning problem in this context is to assign a probability to a measurement signal, such as the frequency of a heartbeat.

Example 2.1.4: Hypothesis guessing

In a notorious assessment test for job interviews, a candidate is given a list of integers between 1 and 100, for example $\{4, 16, 36, 100\}$ and has to 'complete' the series, i.e. find new instances produced by the same rule. In order to do so, the candidate has to guess the rule or hypothesis with which these numbers were randomly generated. One guess may be the rule 'even numbers out of 100' (H1), but one could also think of 'multiples of 4' (H2), or 'powers to the 2' (H3). One intuitive way of judging different hypothesis that all fit the data is to prefer those that have a smaller amount of options, or in this example, that are true for a smaller amount of

[2] See `http//:yann.lecun.com/exdb/mnist`, last accessed Jan 2017.

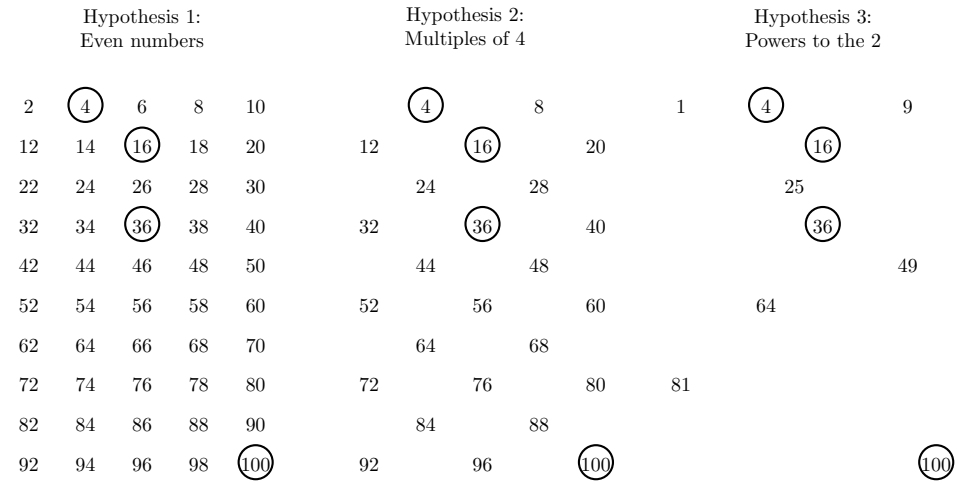


Figure 2.3: Illustration of Example 2.1.4. The circled numbers are given with the task to find a natural number between 1 and 100 generated by the same rule. There are several hypotheses that match the data and define the space of numbers to pick from.

numbers. For example, while H1 is true for 50 numbers, H2 is true for only 25 numbers, and H3 only fits to 10 numbers. It would be a much bigger coincidence to pick data with H1 that also fulfills H3 than the other way around. In probabilistic terms, one may prefer the hypothesis for which generating exactly the given dataset has the highest probability [14]. (Maximum likelihood is an optimisation objective introduced below that is based on exactly this concept).

In these examples, the inputs were given by images, time series, spectra and integers, while the outputs were the content of an image, a price forecast, the chance of a patient being sick, a number from a set. If the training data consists of input-output pairs as in Examples 2.1.1-2.1.3 one speaks of *supervised learning*, while data that does not come with target outputs poses an *unsupervised learning* problem as in Example 2.1.4. A third area of machine learning is *reinforcement learning*, in which an agent gets rewarded or punished for certain decisions according to a given rule, and the agent learns an optimal strategy by trial and error. In this thesis I will focus on supervised learning problems only.

2.1.2 Problem definition of supervised pattern recognition

The basic structure of a supervised pattern recognition task can be formally defined as follows.

Definition 1. ***Supervised pattern recognition problem.*** *Given an input domain $\mathcal{X}$ and an output domain $\mathcal{Y}$, a training data set $\mathcal{D} = \{(x^{(1)}, y^{(1)}), ..., (x^{(M)}, y^{(M)})\}$ of M training pairs $(x^{(m)}, y^{(m)}) \in \mathcal{X} \times \mathcal{Y}$, as well as a new unclassified input $\tilde{x} \in \mathcal{X}$, the task is to guess or predict the corresponding output $\tilde{y} \in \mathcal{Y}$.*

There are closely related tasks that can be understood as variations of supervised pattern recognition which will be discussed here as well. *Pattern completion* does not define distinct outputs, but takes an incomplete input with the task of finding the missing value(s). For *associative memory* a

Input	Output
Regression tasks	
last month's oil price	tomorrow's oil price
search history of a user	chance to click on a car ad
insurance customer details	chance of claiming
Multilabel classification tasks	
images	cat, dog or plane?
recording of speech	words contained in speech
text segment	prediction of next word to follow
Binary classification tasks	
text	links to terrorism?
video	contains a cat?
email	is spam?
spectrum of cancer cell	malicious?

Table 2.1: Examples of supervised pattern classification tasks in real-life applications.

new input-output data point is given, and the output gets 'corrected' if not equal to the prediction.

Note that the above defines a 'classical pattern recognition problem', which means that the data is generated by objects described by classical physics. Formulations of pattern recognition for non-Kolmogorovian structures have been defined [6] and some properties such as nonsignaling principles for inductive learning [20] and Bayesian inference [21] have been investigated, but refer to the previously defined 'QQ' case and are out of the scope of this thesis.

In most applications considered here (and if not stated otherwise) the input domain $\mathcal{X}$ is chosen to be the $\mathbb{R}^N$ or for binary variables, the space of N-bit binary strings $\{0,1\}^N$. The input vectors are also called *feature vectors* as they represent information on carefully selected features of an instance. In cases where the 'raw data' is not in numerical form or does not have an obvious distance measure between the instances - such as words in text recognition- one has to first find a suitable representation. But pre-processing the data is central for any machine learning application and in practice often more important than which model is used for prediction [1]. Important strategies are *feature scaling* such as shifting the data to a zero mean and unit variance, which helps to avoid the unwanted effect of vastly different scales (think for example of the yearly income and age of a person). The raw data can be compressed by *feature extraction* or, in the case of continuous features, they can be expanded by *feature maps* that will be of importance later.

Definition 2. ***Feature map.*** *A feature map is a nonlinear map $\phi : \mathbb{R}^N \to \mathbb{R}^K$ which transforms the input data into another representation. Usually the new feature space is of higher dimension than the original input space, or $K >> N$.*

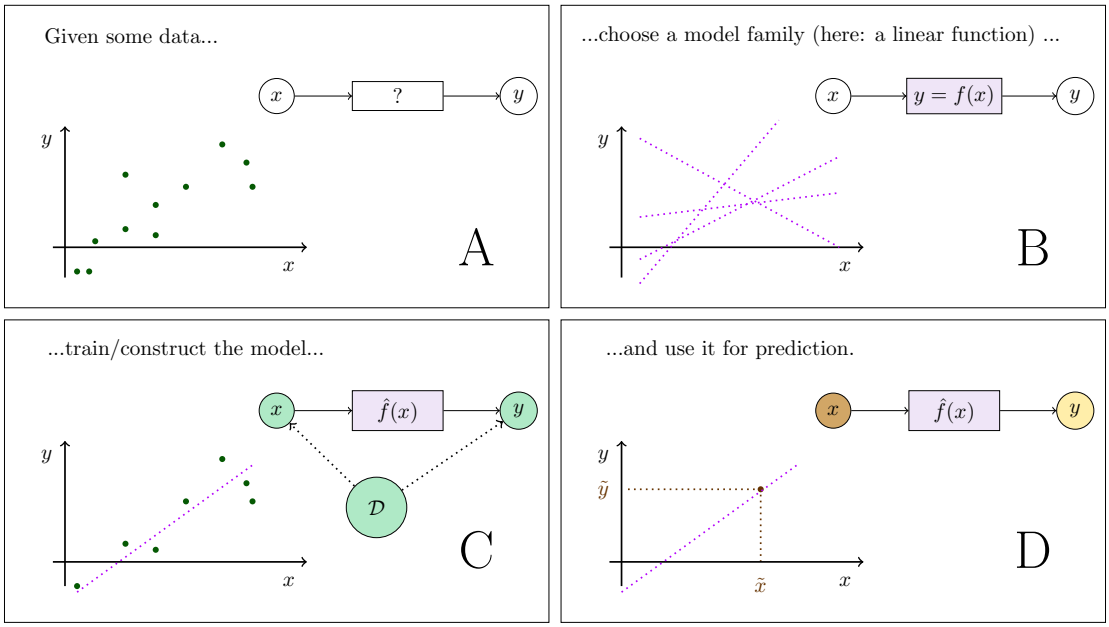


Figure 2.4: The five steps of a machine learning algorithm, here illustrated for a regression problem. (A) A dataset of one-dimensional inputs and outputs is given, produced by an unknown relationship between the two. A method specifies a model family (in this case a linear model) (B), and specifies how to train the model with the training set (C). The model can then be used for prediction (D).

As shown later, feature maps allow us to use simple (i.e., linear) models to explain complicated relationships between data by projecting them into higher dimensional spaces.

The choice of the output domain determines another important distinction in the type of problem to be solved. If $\mathcal{Y}$ is a set of D discrete class labels $\{c_1, ..., c_D\}$ one speaks of a *classification* task. Every D-class classification problem can be converted into $D-1$ *binary classification* problems by successively asking whether the input is in class $c_d, d = 1, ..., D-1$ or not. This is why I will only consider the binary classification problem $\mathcal{Y} = \{0, 1\}$ in the first place. *Regression* refers to problems in which $\mathcal{Y} = \mathbb{R}$ or an interval of the real numbers. Although classification and regression imply two different mathematical structures, most machine learning methods have been formulated for both versions. A classification method can often be generalised to regression by switching to continuous variables and adjusting the functions or distributions accordingly, while the outcome of regression can be discretised (i.e. through interpreting $y > 0 \to 1$ and $y \leq 0 \to 0$).

Examples of inputs and outputs for classification and regression problems can be found in Table 2.1. They might also illustrate why machine learning gains so much interest from industry and governments: Good solutions to any of these problems is worth billions of dollars in military, medical, financial, technical or commercial applications.

2.1.3 How data leads to a predictive model

The term 'model' and its role in machine learning is comparable with the term of a 'state' in quantum mechanics - it is a central concept with a clear definition for those who use it, but it takes a long practice to grasp all dimensions of it. One could define models in machine learning as sets of equations, rules or algorithms that recover the relationship or the map between inputs

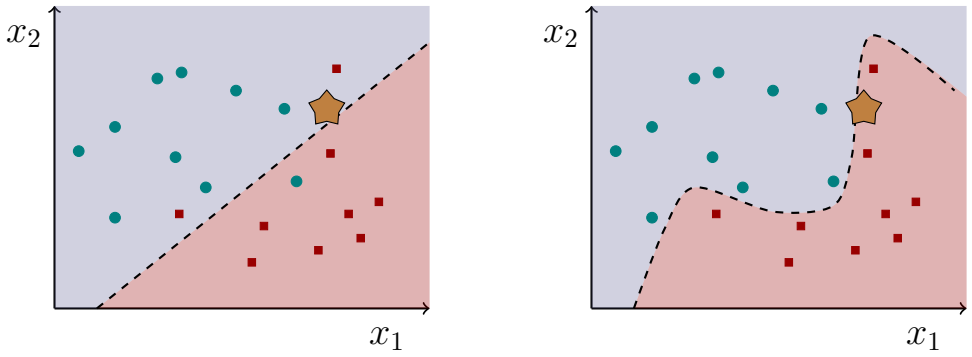


Figure 2.5: Plot of the input space of a classification problem. A model for binary classification divides the input space into regions where it classifies inputs as class 1 (here red rectangles) or class 2 (here blue circles). The flexibility of the decision boundary can change the classification result for a new input (star shape) considerably. In the left case, the new input would be classified as a circle, while the more flexible model assigns it to a region of rectangles.

to outputs. Mathematically speaking, a map $\mathcal{M} : \mathbb{R}^N \to \mathbb{R}$ or $\mathcal{M} : \{0,1\}^N \to \{0,1\}$ is a function. In the next section another option will become apparent, in which the essential ingredient of the map is a probability distribution over the possible results, from which the desired answer can be derived. I will understand a machine learning model therefore always as a function on or as a probability distribution over the input space.

Figure 2.4 shows four steps of how data leads to a predictive model. Given some data, a generic model family has to be chosen. This can for example be a linear function $f(x_1...x_N) = w_1x_1 + ... + w_Nx_N$ with parameters $w_1, ..., w_N$. The model family defines the type of possible functions that can be used. In the case of a classification task, it defines the complexity of the decision region pattern that separates the input space regarding the two classes (see Figure 2.5).[3] The model is then trained or 'adapted' to the data, in other words, a specific model function is chosen from the family. This defines a fixed decision boundary according to how each point in the input space is classified by the model. The training step is the crucial procedure, and two main approaches will be discussed below: Some models depend on parameters which are fitted in a distinct *training phase* after which the data can be discarded and all information is saved in the parameters, while other models explicitly depend on the data. Once the model is adapted to the data it can be used for prediction.

In most cases training means to define an objective function that quantifies the quality of a model, and the problem reduces to finding an optimum to this function. Machine learning tends to translate into rather difficult optimisation problems that require a lot of computational resources. This is why optimisation "lies at the heart of machine learning" [22]. In fact, major breakthroughs in the history of machine learning often came with a new way of solving an optimisation problem. For example, the two influential turning points in neural networks research were the introduction of the backpropagation algorithm in the 1980s [23][4] as well as the use of Boltzmann machines to

[3] Obviously, the choice of a model and the details in which it is implemented can significantly influence the result of the prediction, which is why it requires a lot of experience to get a feeling for which method matches a dataset well. In this sense machine learning will always be a concerted effort between computer and human.

[4] The original paper, a Technical Report by Rumelhart, Hinton and Williams , has close to 20,000 citation on *Google Scholar* at the time of writing. It is widely known today that the algorithm had been invented by others long before this upsurge of attention.

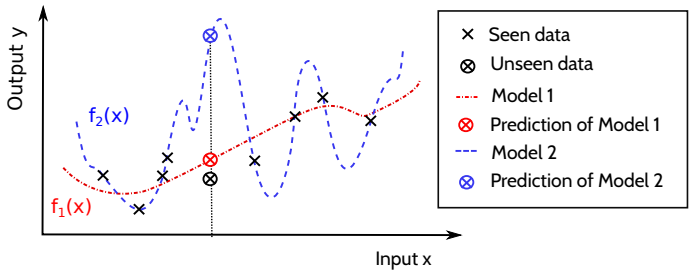


Figure 2.6: Illustration of the principle of overfitting in linear regression. While the blue model function fits the data points perfectly well, it does not give a very accurate prediction for the new input. Meanwhile, the red model recovers the trend in the data a lot better.

train deep neural networks in 2006 [24].

Machine learning research developed a number of approaches to supervised pattern recognition which each define a generic model based on a set of assumptions as well as a way to do the training. I will call these approaches *methods*. Introduced below are amongst others feed-forward neural networks, linear regression or Bayesian nets. Each method comes with a distinct language, mathematical background and its own separate scientific community. It is interesting to note that these methods are not only numerous and full of variations, but remarkably interlinked. One method can often be derived from another even though the two are rooted in very different theoretical backgrounds.

2.1.4 Estimating the quality of a model

A predictor gives good results if the prediction coincides with the 'actual' values. This is obviously a problem since we do not know the 'actual' values at the point of making the prediction. In many problems such as the oil price forecast or the image recognition problem from above the actual value can be determined in future times or by asking for human judgment. In other problems such as decision under uncertainty the best solution might never be clear.

Under these conditions, how can we determine the quality or predictive power of a model? The idea is to use a share of the data itself for evaluation. One divides the dataset into a training set and a test set, and while the model is built based on the training set, its quality is measured by how accurately it can recover the target output values in the test set. The error on the training set is called the *empirical error*, while the error on the test set is regarded as an approximation of the chance of predicting a new input wrong, which is called *generalisation error* or *structural risk* (the counterpart to the error ϵ is the *accuracy* $1 - \epsilon$). An important fact is that a low empirical or training error does not necessarily lead to a low generalisation error. Hence, and counter the first intuition, a good model *is not* necessarily a model that explains the training data perfectly well. Instead, good models have to find a balance between explaining the training dataset sufficiently well, but at the same time not *overfitting* (i.e. learning the particulars of the training set instead of the general structure behind the data). The performance of a machine learning algorithm is therefore measured in its *generalisation ability*.

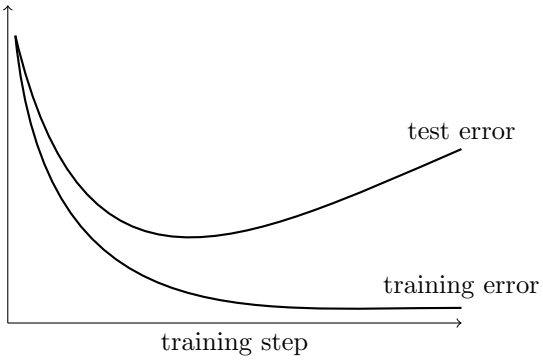


Figure 2.7: Typical behaviour of the error on the training and test set during training. While the model learns to reduce the error on the training set, the test error usually begins to increase after a certain number of training steps, an indicator that the model begins to learn the specific structure of the training set, loosing its ability to generalise. Counter-strategies are early stopping, choosing a less flexible model family or penalising more complex models.

Overfitting and generalisation can be understood when thinking of an experimentalist who wants to recover the law with which her experiment produced some data. One can always find a high-order polynomial that goes through every data point and thus fits the data (here the training set) perfectly well. However, if we look at additional data points produced by the same experiment (the test set), one might discover that the high-order polynomial has large errors in producing the additional data, in other words it does not reflect the general structure of the physical law on which the experiment is based.

The concept of generalisation has been quantified by Vladimir Vapnik and Alexey Chervonenkis in their famous statistical risk minimisation framework [25]. They find that an upper bound for the generalisation error is given by the empirical error plus an expression that depends on the number of data M and an integer d called the VC-dimension of a model. Roughly speaking, the VC-dimension is a measure of how many ways a model offers to separate a dataset into regions of different classes, or how flexible it is.[5] More flexible models (like a high polynomial for regression) usually lead to a lower empirical or test error, but increase the second term in the inequality. As a consequence, while training usually continuously reduces the error (i.e. number of misclassified instances or least squares distance between model and data) on the training set, the error on the test set may increase after a certain number of steps in the training procedure, which is an indicator for overfitting (see Figure 2.7). Strategies to avoid overfitting are called *regularisation*, and can be as simple as 'early stopping' of the training, or to favour small or sparse solutions when estimating the optimal parameter of a model. Another important idea is to choose a model that is not flexible enough to learn the specifics of the training set. Overfitting and regularisation are important concepts in machine learning.

[5] However, beyond a small class of models this parameter is nontrivial to calculate and usually computed by numerical simulations.

Model	nonparametric	parametric
deterministic	$f(x, \mathcal{D})$	$f(x, \theta)$
probabilistic	$p(y, x\|\mathcal{D})$	$p(y, x\|\theta)$

Table 2.2: Distinction between deterministic vs. probabilistic models and parametric vs. nonparametric models.

2.2 Different types of models

In order to have a closer look into models and training strategies, I want to distinguish different types of models along two dimensions (see Table 2.2). The first dimension separates *deterministic* and *probabilistic* methods. The mathematical object defining a deterministic model is a function with a predefined output for each input. Probabilistic models define probability distributions over data points and derive predictions from those distributions. In both cases models (i.e., the function or probability distribution) can be *parametric* or *nonparametric*, which is the second dimension. Parametric models are functions or distributions that are parametrised. The number of parameters together with the flexibility of the model is fixed beforehand (see also Figure 2.8). Nonparametric models sometimes 'consult' the training data directly when making a prediction (see Figure 2.9), and parameters are used to determine which training input is more important than others. The number of these parameters and thereby the flexibility of the model grow with the size of the training set.[6]

2.2.1 Deterministic approach

The deterministic approach solves the supervised pattern classification or regression problem outlined in Definition 1 by using the dataset $\mathcal{D}$ to find a *model function* $f : X \to Y$. A prediction is the evaluation of the function for a certain input, $y = f(x)$.

2.2.1.1 Parametric model functions

In parametric methods, f is a model family that depends on some free parameters $\theta \in \Theta$. The model function relating inputs to outputs is therefore of the form

$$y = f(\mathbf{x}; \theta). \tag{2.1}$$

Note that in the following these parameters will sometimes be expressed by a weight vector $\mathbf{w} = (w_1, ..., w_N)^T \in \mathbb{R}^N$.

Learning translates to estimating the 'best' parameters $\hat{\theta}$ given the data. The notion of 'best' ultimately refers to a low generalisation error of a model (2.1) and as explained in the last section, this means to lower the empirical error on the training data set while not overfitting. To quantify this, an *objective function* or *loss function* mapping from the parameter space to the real

[6] Again, such distinctions are not sharp as parametric models in the so called dual formulation can become nonparametric, and nonparametric models can have a fixed number of hyper-parameters that can be learned from the data; deterministic and probabilistic procedures can be mixed up when introducing noise into the former, or taking mean values of the latter.

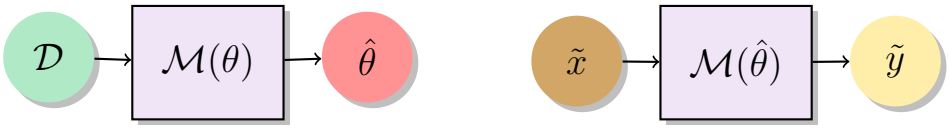


Figure 2.8: Schematic illustration of the learning (left) and prediction (right) phase of parametric machine learning methods. Given a model $\mathcal{M}(\theta)$ with free parameters θ and a data set $\mathcal{D}$, learning means to estimate an optimal set of parameters $\tilde{\theta}$ that generalises from the data. The trained model $\mathcal{M}(\hat{\theta})$ maps a new input $\tilde{x}$ to the predicted output $\tilde{y}$.

numbers, $o : \Theta \to \mathbb{R}$, is defined. The training procedure minimises the function, thereby solving an optimisation problem. Regularisation can be included into the optimisation problem via constraints (i.e., to force the parameters to be sparse or small), or as part of the solution method (i.e., stopping before the global minimum is found, or setting small weights in the solution to zero).

A very common choice for the objective function $o(\theta)$ is the least squares error which compares the outputs $f(\mathbf{x}^m, \theta)$ produced by the model when fed with the inputs of the training data set with the target outputs in the data set y^m,

$$o(\theta) = \sum_{m=1}^{M} |f(\mathbf{x}^m, \theta) - y^m|^2, \tag{2.2}$$

The resulting optimisation problem is known in statistics as *least-squares optimisation*:

Definition 3. ***Least-Squares Optimisation.*** *Given a model $f : \mathcal{X} \times \theta \to \mathcal{Y}$ with suitable input, output and parameter domains $\mathcal{X}, \mathcal{Y}, \theta$, as well as a dataset $\mathcal{D}$ with tuples $(\mathbf{x}^m, y^m) \in \mathcal{X} \times \mathcal{Y}$. Find*

$$\hat{\theta} = \min_{\theta} \sum_{m=1}^{M} |f(\mathbf{x}^m, \theta) - y^m|^2. \tag{2.3}$$

Regularisation constraints can be included by adding a 'penalty' term $\lambda||\hat{\mathbf{w}}||$ with strength controlled by λ to the right side. Different norms $||\cdot||$ favour different solutions, for example sparse or short vectors.

Mathematical optimisation theory developed an extensive framework to classify and solve optimisation problems [26] which are called *programmes*, and there are important distinctions between types of programmes that roughly define how difficult it is to find a global solution with a computer (For some problems, even local or approximate solutions are hard to compute). The most important distinction is between convex problems for which a number of algorithms and extensive theory exists, and nonconvex problems that are a lot harder to treat [27]. Convexity thereby refers to the objective function and possible constraint functions. Roughly speaking, a set is convex if a straight line connecting any two points in that set lies inside the set. A function $f : \mathcal{X} \to \mathbb{R}$ is convex if $\mathcal{X}$ is a convex domain and if a straight line connecting any two points of the function lies 'above' the function (for more details see [26]).[7] To give an example, least-squares optimisation in

[7] Two reasons explaining why convex optimisation is relatively well understood is that a) they only have global optima, and b) local information on the function contains global information as well. An optimisation algorithm can therefore better determine the search direction and does not risk to get stuck in local minima.

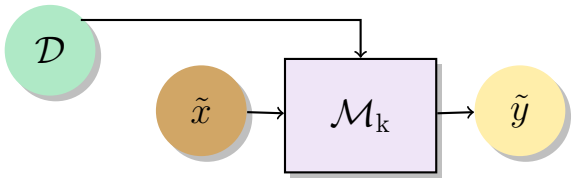


Figure 2.9: Schematic illustration of similarity based machine learning methods. The model (or often a kernel as part of the model, here indicated by k) is 'directly' constructed from the dataset $\mathcal{D}$ and used for prediction.

Definition 3 based on a model function that is linear in the parameters is a rather simple convex quadratic optimisation problem that has a closed-form solution. For general nonconvex problems much less is known, and many machine learning problems fall into this category. Popular methods are therefore iterative searches such as *gradient* or *steepest descent*, which performs a stepwise search for the minimum.

Box 2.2.1: Gradient descent method

In gradient descent methods the parameters θ of an objective function $o(\theta)$ are successively updated according to

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla o(\theta^{(t)}), \tag{2.4}$$

where η is an external parameter called the *learning rate*. The gradient nabla $o(\theta^{(t)})$ always points towards the ascending direction in the landscape of o, and following its negative means to descend into valleys. As one can imagine, this method can get stuck in local minima if they exist, and convergence to a minimum can take a long time. The advantage is its simplicity and applicability in many settings. Note that there are many variations of gradient descent that improve on the basic update. An important variation is stochastic gradient descent, where small alternating batches of the training set are evaluated in the objective function. Another improvement comes from a step-dependent learning rate $\eta^{(t)}$.

2.2.1.2 Nonparametric model functions

Nonparametric model functions do not depend on a fixed number of variables. I will particularly look at so called "kernel" methods, which use the data points themselves for classification. The more data, the more flexible the model becomes. An important idea for kernel methods is that "similar inputs have similar outputs" [18] and their heart is usually a similarity measure on the input space $\mathcal{X}$. A simple example is the k-nearest neighbour method. Given a new input, the prediction is the average or majority output amongst the k closest training inputs. Similarity can here be defined by the Euclidean distance.

Parametric models can sometimes be turned into nonparametric ones. For example, for a (parametric) linear model of the form $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T\mathbf{x}$ one can assume the weight vector to be a linear combination of training inputs $\mathbf{w} = \sum_m \alpha_m \mathbf{x}^m$ and get the (nonparametric) classifier $f(\mathbf{x}; \mathbf{w}) = \sum_m \alpha_m (\mathbf{x}^m)^T\mathbf{x}$. Typically, inner products between the training inputs and the new

input appear. The so called *kernel trick* allows us to give such models a lot of power and flexibility.

Box 2.2.2: The kernel trick

In machine learning, kernels are defined as follows:

Definition 4. ***Kernel.*** *A kernel is a bivariate function* $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *such that for any set* $\{x_1, ..., x_N\} \subset \mathcal{X}$ *the matrix* $\mathbf{K}$ *called kernel or Gram matrix with entries*

$$K_{ij} = \kappa(x_i, x_j), \quad x_i, x_j \in \mathcal{X} \tag{2.5}$$

is positive (semi-)definite. As a consequence, $\kappa(x_i, x_i) > 0$ *and* $\kappa(x_i, x_j) = \kappa(x_j, x_i)$.

The scalar product between two real vectors is a kernel.

The kernel trick can be expressed as follows:

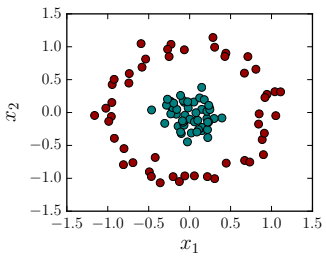
> Given an algorithm which is formulated in terms of a positive definite kernel κ, one can construct an alternative algorithm by replacing κ by another positive definite kernel κ'. [28]

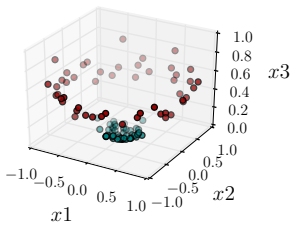
This becomes interesting when Mercer's theorem states that every kernel can be expressed as a scalar product,

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle ,$$

where $\phi : \mathcal{X} \to \mathcal{X}'$ maps the input vectors into a (usually higher dimensional) feature space on which a scalar product is defined [29].

The consequences are important. For every positive semi-definite kernel $\kappa(x_i, x_j)$ there is a feature map (see Definition 2) so that $\kappa = \langle \boldsymbol{\phi}(x_i), \boldsymbol{\phi}(x_j) \rangle$. In other words, given a machine learning model in which the inputs appear in form of a kernel (for example a simple dot product between input and training inputs), replacing the kernel by another kernel *effectively* can implement a nonlinear feature map into a higher dimensional space. In this higher dimensional space the data can often be processed with much simpler models: Consider a dataset of two concentric circles is impossible to separate by a linear decision boundary, the feature map $\phi((x_1, x_2)^T) = (x_1, x_2, 0.5(x_1^2 + x_2^2))^T$ transforms it into a linearly separable dataset.





Note that the feature map can also map into spaces of infinite dimension. The crux is that one never has to calculate the scalar product in this space, but simply calculates the kernel function with the original inputs.

As an example, take the squared exponential kernel function with $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^N$ and use the series expansion of the exponential function to get

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{1}{2}|\mathbf{x}_i - \mathbf{x}_j|^2} = \sum_{j=0}^{\infty} \frac{(\mathbf{x}_i^T \mathbf{x}_j)^k}{k!} \mathrm{e}^{-\frac{1}{2}|\mathbf{x}_i|^2} \mathrm{e}^{-\frac{1}{2}|\mathbf{x}_j|^2} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \,.$$

The squared exponential kernel effectively implements a feature map $\phi(\mathbf{x}) = (\mathrm{e}^{-\frac{1}{2}|\mathbf{x}_i|^2}, \mathbf{x}\mathrm{e}^{-\frac{1}{2}|\mathbf{x}_i|^2}, \mathbf{x}^2\mathrm{e}^{-\frac{1}{2}|\mathbf{x}_i|^2}, ...)^T$ into an infinite dimensional space.

Excellent introductions to kernel methods are given by Refs. [30, 28].

A number of kernel methods will be introduced in the next Section.

2.2.2 Probabilistic approach

Machine learning, as does statistics, has to deal with uncertainty, and a lot of methods can be formulated or reformulated in a probabilistic language or as *statistical models* (as demonstrated in the textbook of Murphy [14]).

The general idea of the probabilistic approach is to understand the inputs and outputs $(\mathbf{x}, y)$ as random variables drawn from a joint probability distribution $p(\mathbf{x}, y)$, which will be called the *model distribution* here. Figure 2.10 shows the example of a one-dimensional regression problem $(\mathcal{X}, \mathcal{Y} = \mathbb{R})$. The black points are data samples drawn from an unknown distribution. Given such data, the goal is to find a model from which it would have most likely been drawn. Here this is a bivariate normal distribution. Once the joint model distribution is known, one can form the *class-conditional probability* $p(y|\mathbf{x}) = p(\mathbf{x}, y)/p(\mathbf{x})$, which effectively cuts through the joint distribution for a certain input as indicated by the black line. There are two common practices [31] to obtain the desired output y from $p(y|\mathbf{x})$: The *Maximum A Posterior estimate* chooses the output for which the class-conditional probability is maximised,

$$y = \max_y p(y|\mathbf{x}), \tag{2.6}$$

while an alternative is to take the mean of the distribution,

$$y = \int p(y|\mathbf{x})\; y \; dy, \tag{2.7}$$

which in classification tasks reduces to a sum.

While *generative* probabilistic models follow this strategy and derive the full distribution from the data, *discriminative models* directly try to obtain the slimmer class-conditional distribution $p(y|\mathbf{x})$. Given the so called likelihood distribution of the inputs for a given output, $p(\mathbf{x}|y)$, one can infer

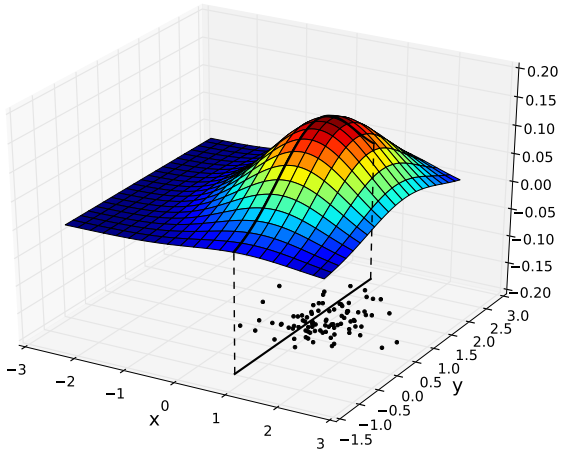


Figure 2.10: Probabilistic models treat inputs and outputs as random variables drawn from a probability distribution. To use the distribution for prediction, one derives the marginalised distribution over the outputs given the input (here indicated by the black line intersecting with the distribution) and estimates its maximum. This is of course also possible for discrete inputs and outputs, where the distribution can be displayed in a table listing all possible combinations of the random variables.

the class-conditional distribution via Bayes' theorem. Even though generative models contain a lot more information and it may seem that they need much more resources to be derived from data, it is by no means clear which type of model is better and as so often depends on the task at hand [32].

2.2.2.1 Parametric probability distributions

Similar to the deterministic case, probability distributions can of course be parametrised (indicated here by $p(\mathbf{x}, y; \theta)$ for the joint model distribution and $p(y|\mathbf{x}; \theta)$ for the class conditional model distribution). Examples for parametrised distributions in one dimension are the normal or Gaussian distribution with θ consisting of mean μ and variance σ^2,

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

as well as the Bernoulli distribution for binary numbers with $\theta = q$, $0 \leq q \leq 1$,

$$p(x; q) = q^x (1-q)^{1-x}.$$

Like in the deterministic case, learning reduces to estimating a set of parameters θ which fully determine the model distribution.

In statistics, estimating the parameters of a probabilistic mathematical model given a dataset is often done through *maximum likelihood estimation*. In fact, least squares can be shown to be the maximum likelihood solution under the assumption of Gaussian noise [12]. The underlying idea is to find parameters θ so that there is a high probability that the data have been drawn from the distribution $p(\mathbf{x}, y; \theta)$. This logic has already been encountered in Example 2.1.4.

Definition 5. ***Maximum likelihood estimation***
Given a probability distribution $p : \mathcal{X} \times \mathcal{Y} \times \theta \to [0, 1]$ *with suitable input, output and parameter domains* $\mathcal{X}, \mathcal{Y}, \theta$*, as well as a dataset* $\mathcal{D}$ *of tuples* $(\mathbf{x}^m, y^m) \in \mathcal{X} \times \mathcal{Y}$*. Find*

$$\hat{\theta} = \max_{\theta} \sum_{m=1}^{M} p(\mathbf{x}^{(m)}, y^{(m)}; \theta). \tag{2.8}$$

It is standard practice to insert a logarithm into Eq. (2.8) as it does not change the solution of the optimisation task while giving it favourable properties.

The rule for maximum likelihood estimation can be derived beautifully using the framework of Bayesian probability calculus. Bayes' famous rule derived from probability theory is given by

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)}, \tag{2.9}$$

where a,b are values of (sets of) random variables A,B, and $p(a|b)$ is the conditional probability of a given b defined as

$$p(a|b) = \frac{p(a, b)}{p(b)}.$$

The term $p(b|a)$ in Eq. (2.9) is called the *likelihood*, $p(a)$ is the *prior* and $p(a|b)$ the *posterior*. The application of Bayesian probability theory to machine learning problems is called 'Bayesian learning' [33, 13]. Box 2.2.3 briefly introduces into the logic behind Bayesian learning; not only to show how to obtain maximum likelihood optimisation but also as an important foundation to later chapters.

Box 2.2.3: Bayesian Learning

Given a training dataset $\mathcal{D}$ as before and understanding $\mathbf{x} = (x_1, ..., x_n)$ as well as y as random variables, we want to find the probability density $p(\mathbf{x}, y|\mathcal{D})$. In parametrised methods, the distribution $p(\mathbf{x}, y)$ is fully determined by a set of parameters θ, and formally one can write

$$p(\mathbf{x}, y|\mathcal{D}) = \int p(\mathbf{x}, y; \theta|\mathcal{D}) d\theta = \int p(\mathbf{x}, y|\theta) p(\theta|\mathcal{D}) d\theta. \tag{2.10}$$

The first part of the integrand, $p(\mathbf{x}, y|\theta)$, is given by the model distribution $p(\mathbf{x}, y; \theta)$ that one assumes. We therefore need to find $p(\theta|\mathcal{D})$. Using Bayes' formula reveals

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta}. \tag{2.11}$$

The prior $p(\theta)$ describes our knowledge on which parameters lead to the best model *before* seeing the data. For example, one could find it more likely that a coin is fair than that it always lands on one side. If nothing is known, the prior can be chosen uniform. The likelihood $p(\mathcal{D}|\theta)$ is the probability of seeing the data given certain parameters. Together

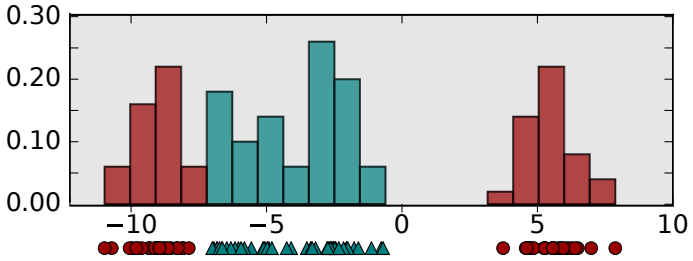


Figure 2.11: A histogram derived from one-dimensional inputs of two different classes (blue triangles and red circles) can be used to derive coarse-grained class-conditional probability distribution over the data.

with the normalisation factor $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$ we get the posterior $p(\theta|\mathcal{D})$ which is the probability of parameters θ *after* seeing the data.

The equation shifts the problem to finding the distribution over the data given the parameters. Under the assumption that the data samples are drawn independently, one can factorise the distribution and write

$$p(\mathcal{D}|\theta) = \prod_m p(\mathbf{x}^{(m)}, y^m|\theta).$$

To chose the most likely parameters given the data, we have to maximise this expression over θ, which is exactly the maximum likelihood optimisation problem in Definition 5.

An interesting consideration about the computational complexity arises from Bayesian learning. If one could compute the full integral in Equation (2.10), one would not have to do maximum likelihood optimisation [33]. Computing the integral can easily become prohibitive. It seems that optimisation and integration are two sides of the same coin, and both are hard problems to solve when no additional structure is given.

2.2.2.2 Nonparametric probability distributions

Nonparametric statistical models are probability distributions that do not depend on a fixed amount of parameters, but on the data themselves. A very simple way to construct nonparametric probability distributions from data are histograms. As shown in Figure 2.11 one simply counts the number of data in a certain interval Δx_i of the input space. The probability distribution would then be the amount of data in that interval M_i divided by the total amount of data,

$$p(\Delta x_i) = \frac{M_i}{M}.$$

The smooth version of this is called a kernel density estimator and will be introduced below. Of course, kernel tricks are also applicable to statistical models.

2.3 Important methods of machine learning

As a summary of the previous sections, a method in machine learning specifies a *problem structure* (the input and output spaces), a general ansatz for a *model function or distribution* that can be

used for prediction, and a *training method* of how to use the data to construct a specific model that generalises from the data. In the remainder of the chapter, some important machine learning methods will be discussed. The models they define as well as popular training strategies are summarised in Table 2.3 as an overview or for quick reference.[8]

2.3.1 Data fitting

Most physicists are familiar with statistical methods for data fitting such as *linear* and *nonlinear* regression.[9] These are well-established in statistics and data science but also play an important role in machine learning, illustrating the proximity of the two fields.

2.3.1.1 Linear Regression

Needless to say, linear regression tackles the *problem of regression* outlined in Definition 1 and is based on a deterministic linear model function (see Eq. (2.1)),

$$f(\mathbf{x};\mathbf{w}) = \mathbf{x}^T\mathbf{w} + w_0, \tag{2.12}$$

where the vector $\mathbf{w} \in \mathbb{R}^N$ contains the parameters. The bias w_0 can be included into $\mathbf{x}^T\mathbf{w}$ by adding an extra variable $x_0 = 1$ and will therefore be neglected in the following. Note that the term 'linear' refers to linearity in the model parameters only. A nonlinear feature map on the original input space can turn linear models into powerful predictors that can very well be used to model nonlinear functions. A well known example is a feature map $x \in \mathbb{R}$, $\phi : \mathbb{R} \to \mathbb{R}^{d+1}$ with $\phi(x) = (1, x, x^2, ..., x^d)^T$, so that f in Eq. (2.12) becomes

$$f(\phi(x);\mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_d x^d \ .$$

According to the Weierstrass approximation theorem [34], any real single-valued function that is continuous on a real interval $[a, b]$ can be arbitrarily closely approximated by a polynomial function. In practice, this might involve an infinite number of parameters w_i, and other methods such as nonlinear regression explained below, or more intricate feature maps might be much more elegant and simple.

Learning in linear regression means to find the parameters $\mathbf{w}$ that fit f to the data in order to predict new data points. A very successful[10] approach to find the optimal parameters is *least squares estimation* introduced in Definition 3. The objective function in (2.3) becomes

$$o_{\mathrm{lr}}(\theta) = \sum_{m=1}^{M} |(\mathbf{x}^{(m)})^T\mathbf{w} - y^m|^2. \tag{2.13}$$

In matrix form this reads

$$o_{\mathrm{lr}}(\theta) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}),$$

[8] The categorisation of different methods into *data fitting*, *neural networks*, *graphical models* and *kernel methods* is not unambiguous: Boltzmann machines are graphical models as much as neural networks are a type of nonlinear regression; Gaussian processes and support vector machines can be derived from linear models. However, the typology serves as a starting point that captures many crucial points of machine learning and facilitates the discussion of basic concepts in later sections about quantum machine learning methods.

[9] In this context the term regression is both used for the problem of regression as well as the method or model.

[10] Least squares can be shown to produce an unbiased estimator, $E[\hat{\mathbf{w}}] - \mathbf{w} = 0$, with minimum variance [19].

Method	Model function/distribution	Training
Data fitting		
Linear regression	$f(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x}$	convex LS $\rightarrow$ closed-form solution requires matrix inversion
Nonlinear regression	$f(\mathbf{x};\mathbf{w}) = \varphi(\mathbf{w},\mathbf{x})$	
Artificial neural networks		
Perceptron	$f(\mathbf{x};\mathbf{w}) = \varphi(\mathbf{w}^T\mathbf{x})$	perceptron training rule
Feed-forward NN	$f(\mathbf{x};\mathbf{W}_1,\mathbf{W}_2,...) = ...\boldsymbol{\varphi}_2(\mathbf{W}_2\boldsymbol{\varphi}_1(\mathbf{W}_1\mathbf{x}))...$	nonconvex LS $\rightarrow$ steepest descent ("backpropagation")
Recurrent NN	$f(\mathbf{x}^t;\mathbf{W}) = \varphi(\mathbf{w}^T f(\mathbf{x}^{t-1};\mathbf{W}))$ with $\mathbf{x}^{t=0} = \mathbf{x}$	nonconvex LS $\rightarrow$ steepest descent/"backpropagation in time"
Boltzmann machine	$p(\mathbf{s};\mathbf{W}) = \frac{1}{Z}\sum_h \; e^{-\sum_{ij} w_{ij}\; s_i s_j}$	MLE $\rightarrow$ sampling/"contrastive divergence" (heuristic)
Graphical models		
Bayesian network	$p(\mathbf{s}) = \prod_{i=1}^{N} p(s_k\|\pi_k;\theta)$	MLE
Hidden Markov model	$p(V,O) = p(v^0)\prod_{t=2}^{T} p(v^t\|v^{t\text{-}1})\prod_{t=1}^{T} p(o^t\|v^t)$	Forward-Backward Algorithm (heuristic)
Kernel methods		
Kernel density estim.	$p(\mathbf{x}\|y=c) = \frac{1}{M}\sum_{m=1}^{M}\kappa(\mathbf{x}-\mathbf{x}_c^{(m)})$	Choose kernel
K-nearest neighbour	$p(\mathbf{x}\|y=c) = \frac{\#\mathrm{NN}_c}{k}$	Choose hyperparameter k
Support vector mach.	$f(\mathbf{x}) = \sum_{m=1}^{M} a_m y^{(m)}\kappa(\mathbf{x},\mathbf{x}^{(m)}) + w_0$	Learn Lagrangian parameters a_m (minimise dual of minimum-margins problem)
Gaussian process	$p(y\|\mathbf{x}) = \mathcal{N}\big[\boldsymbol{\kappa}^T\mathbf{K}^{-1}\mathbf{y};\kappa-\boldsymbol{\kappa}^T\mathbf{K}^{-1}\boldsymbol{\kappa}\big]$	Choose kernel function (i.e., through MLE)

Table 2.3: Summary of the model functions/distributions and important training strategies of machine learning methods presented in this section. LS stands for least squares optimisation and MLE stands for maximum likelihood estimation. In some cases the input and output variables have been summarised to $\mathbf{s} = (s_1,...,s_N)$. Further notation has to be retrieved from the text.

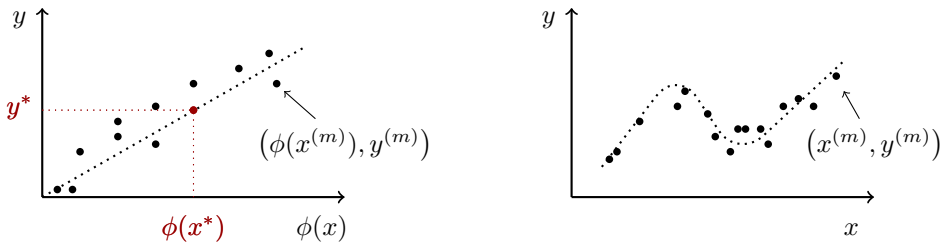


Figure 2.12: Illustration of linear regression with a feature map. The linear model can fit data well in feature space (left) while in the original space the data suggests a nonlinear function (right).

with notation

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(M)} \end{pmatrix}, \qquad \mathbf{X} = \begin{pmatrix} x_1^{(1)} & \dots & x_N^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(M)} & \dots & x_N^{(M)} \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_N \end{pmatrix}.$$

If $\mathbf{X}^T\mathbf{X}$ is of full rank, the estimated parameter vector can be calculated by the closed-form equation

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.14}$$

In computational terms, training a linear regression model reduces to the inversion of a $\mathbb{R}^{M\times D}$ matrix, where usually $D \geq I$, the dimension of the feature space is larger than the dimension of the input vectors. The fastest classical algorithms take time $\mathcal{O}(D^\delta)$, where for the best current algorithms $2 \leq \delta \leq 3$. Various quantum algorithms for linear regression exist and will be introduced later.

2.3.1.2 Nonlinear regression

While in linear regression the model function has a linear dependency on the parameters, nonlinear regression relaxes this condition. One way of deriving nonlinear regression from the linear version is by making the nonlinear feature map ϕ in Eq. (2.12) dependent on the parameters,

$$f(\mathbf{x}) = \phi(\mathbf{w}, \mathbf{x}), \tag{2.15}$$

but ϕ can of course be any nonlinear function φ beyond the concept of a feature map. The most successful nonlinear regression models in machine learning are neural networks.

2.3.2 Artificial neural networks

From a mathematical perspective, neural networks can be seen as a nonlinear regression model with a specific choice for the model function in Eq. (2.15) [17], namely where the input to the nonlinear function is given by a linear model $\mathbf{w}^T\mathbf{x}$. Historically these models were derived from biological neural networks [35, 36] and they have a beautiful graphical representation reminiscent of neurons that are connected by synapses. The nonlinear function originally corresponded to

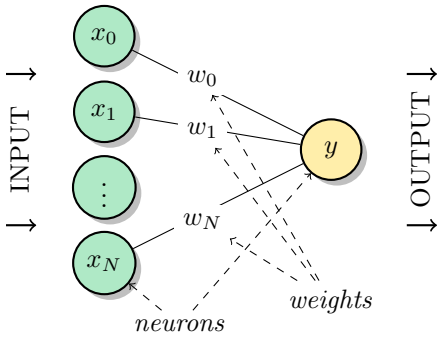


Figure 2.13: Illustration of the perceptron model. The input features are each displayed as nodes of a graph, called units or neurons. The input units are connected to an output unit by weighed edges. Together with an activation function for the output unit, this graph defines a model function $y = \varphi(\mathbf{w}^T\mathbf{x})$.

the 'integrate and fire' principle found in biological neurons [37]. Neural network research was abandoned and revived a number of times during history, and important milestones were when Hopfield showed in 1982 that a certain type of network recovers properties of associative memory [36], the rediscovery of the backpropagation algorithm in the late 80s [38][11], as well as recent developments in 'deep' neural network structures beating several benchmarks in machine learning research [24].

2.3.2.1 Perceptrons

Perceptrons are the basic building block of artificial neural networks. Their model function is given by

$$f(\mathbf{x};\mathbf{w}) = \varphi(\mathbf{w}^T\mathbf{x}), \tag{2.16}$$

where the inputs and weight parameters $\mathbf{w} = (w_1, ..., w_N)^T$ are real numbers, and for convenience the inputs and outputs are often chosen to be either -1 or 1. The nonlinear function φ is called an *activation function* and in the original proposals it referred to the sign or step function

$$\mathrm{sgn}(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ -1, & \text{else.} \end{cases}$$

The perceptron model can be trained using the basic updating rule

$$w_i^{(t+1)} = w_i^{(t)} + \eta(y^{(m)} - \varphi(\mathbf{x}^T\mathbf{w}))x_i^{(m)},$$

where η is the learning rate. Interesting enough, the rule looks very much like a steepest descent method for a linear activation function, optimising the least squares objective $o_{ls} = |\mathbf{w}^T\mathbf{x}^{(m)} - y^{(m)}|^2$ for each training data point separately. The computational properties of a perceptron have been studied since as early as the 1960s [37, 39], showing that the learning rule

[11] Backpropagation was initially developed by Paul Werbos in 1974, during a time in which expert systems were a lot more popular than neural networks, and therefore received little attention.

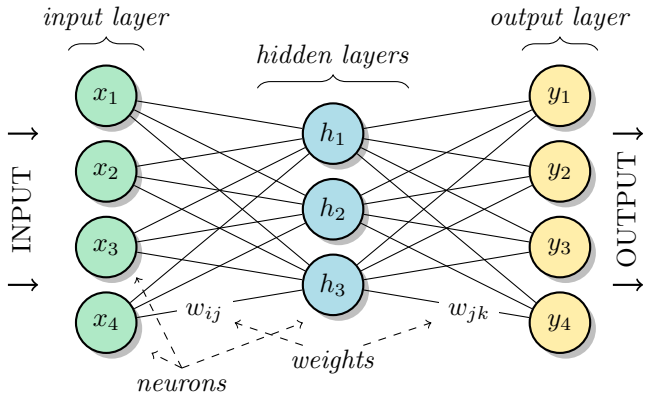


Figure 2.14: A feed-forward neural network (here with only one hidden layer) has only connections between layers.

always converges to the optimal weights. However, after the initial excitement it was found that perceptrons can only learn linearly separable datasets[12], excluding a simple XOR gate from its scope. Only when perceptrons were combined to more complex structures their power becomes apparent, and the perceptron model is the core unit of artificial neural networks.

Figure 2.13 shows a perceptron in the typical graphical representation of neural networks, in which inputs and outputs are understood as units with certain values that are updated by to the units that feed into them. The connections between units are associated with a weight. An activation function is defined for the output node.

2.3.2.2 Feed-Forward Neural Networks

Feed forward neural networks have a model function of the form

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, ...) = \dots \varphi_2 \left(\mathbf{W}_2 \, \varphi_1(\mathbf{W}_1 \mathbf{x}) \right) \dots \quad . \tag{2.17}$$

Here $\varphi_1, \varphi_2, ...$ are nonlinear maps or activation functions between appropriate spaces $\mathbb{R}^{K_1}, \mathbb{R}^{K_2}$.... The parameters are summarised in weight matrices $\mathbf{W}_i$, $i = 1, 2, ...$. The outermost activation function has to map onto the output space $\mathcal{Y}$ which is here always assumed to be one-dimensional, but for simplicity the matrix notation is used for all sets of weights.

The model function is a concatenation of 'linear models' and activation functions, and as the dots suggest, the concatenation can be repeated many times. An interesting perspective on the success of neural networks is that they combine the flexibility of nonlinear dynamics with the data processing power of linear algebra.[13] An important existence theorem by Hornik, Stincombe and White from 1989 [40] proves that only one concatenation of the form

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2) = \mathbf{W}_2 \, \varphi_1(\mathbf{W}_1 \mathbf{x})$$

suffices to make the model a universal function approximator, meaning that up to finite precision it can be used to express any function on a compact domain. This might however require weight

[12] A dataset is linearly separable if it can be divided by a hyperplane in input space.
[13] Private conversation of Hans-Georg Zimmermann, head of Siemens' neural network research group.

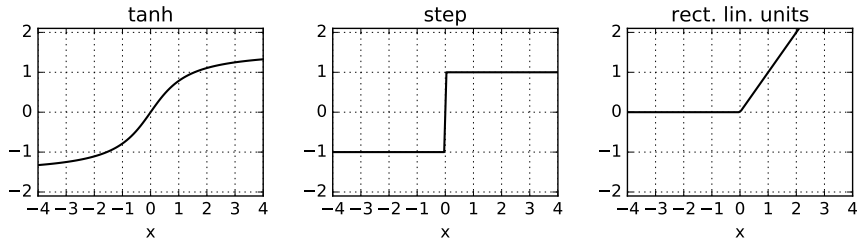


Figure 2.15: Different options for activation functions for input interval $x \in [-4, 4]$: A tangens hyperbolicus, a step function and rectified linear units.

matrices of infinite dimensions.

In terms of their graphical representation, feed-forward neural networks connect multiple perceptrons in layers so that the units of each layer are only connected to the units of the following layer (see Figure 2.14). The first layer is made up of the input units $x_1, ..., x_N$, the following L layers contain the hidden units $h_1^l, ..., h_J^l$ (where $l = 1, ..., L$) whose values are the results of the activation functions in Equation 2.17. The last layer contains the output unit(s). Each neuron is updated by an activation function depending on all neurons feeding into it, and the update protocol prescribes that each layer is updated after the previous one. This way, information is 'fed forward', and an input fed into the first layer is mapped onto an output that can be read out from the last layer after each layer has been updated once. The model for a single hidden layer feed-forward neural network hence reads

$$f(\mathbf{x}; \mathbf{W}_1, \mathbf{W}_2) = \varphi_2(\ \mathbf{W}_2\ \varphi_1(\ \mathbf{W}_1\mathbf{x}\ )\ ). \tag{2.18}$$

A feed-forward neural network has several popular choices of activation functions, which are highly influenced by the choice of the training algorithm. Common examples for these functions are plotted in Figure 2.15. Besides being nonlinear (for at least one layer, otherwise the weights can be summarised to a linear transformation), activation functions are often monotonously increasing and saturating at the extremes such as *tanh* and the logistic function. An important advantage of these two is that their derivative is easy to calculate when working with gradient descent training methods. However, in the 1990s, radial basis functions such as Gaussians were also heatedly debated candidates for activation functions, and more recently *rectified linear units* became popular.

The most common objective function for the training of feed-forward neural networks is the least squares function from Definition 3, which for a single hidden layer reads

$$o(\mathbf{W}_1, \mathbf{W}_2) = \sum_{m=1}^{M} |\varphi_2(\mathbf{W}_2^T\ \varphi_1(\mathbf{W}_1\mathbf{x}^{(m)})) - y^{(m)}|^2. \tag{2.19}$$

For nonlinear activation functions this is in general a nonconvex, nonlinear (and hence very difficult) optimisation problem. By far the most common training algorithm for feed-forward neural networks is therefore an iterative method called *backpropagation* which is based on gradient descent. The weights are incrementally updated with step size or learning rate η so that they descent in the landscape shaped by $o(\mathbf{W}_1, \mathbf{W}_2)$,

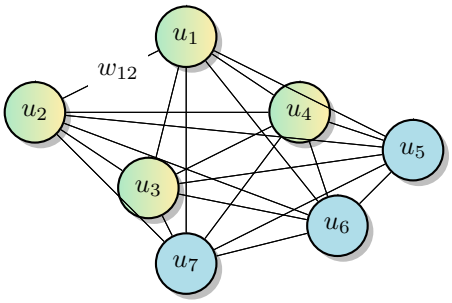


Figure 2.16: A recurrent neural network is represented by an all-to-all connected graph. The same units are used to represent inputs, and at a later time, the outputs (green and yellow nodes). The hidden units (blue) are 'not accessible' and can be understood as pure computational units, adding to the complexity of the dynamics of a recurrent neural net. The network here follows the design of a Hopfield model which does not have self-connections of nodes and where connections have symmetric weights.

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial o(\mathbf{W}_1, \mathbf{W}_2)}{\partial w_{ij}} \tag{2.20}$$

until some convergence criteria is fulfilled. The computational steps needed for one iteration in backpropagation grow polynomial with the number of connections $NJ + JK$ (where K is the number of output units) [16] which can be costly for larger architectures. Convergence times can likewise be prohibitive. A third problem is the large number of parameters, leading to many local and sometimes global minima in the optimisation landscape.

2.3.2.3 Recurrent Neural Networks

While feed-forward neural networks are organised in layers, the graphical representation of recurrent neural networks is an all-to-all connected graph of units which are collectively updated in discrete time steps. Information is therefore not fed-forward through layers, but 'in time'. The input can therefore be understood as the state that some units are set to at time $t = 0$, while the output can be read from one or more designated units at time T. The units that are neither fed with input nor used to read out the output are called 'hidden units' and have a mere computational purpose.

Let $\mathbf{s}^t = (s_1^t, ..., s_K^t)^T$ describe the state of the units s_i, $i = 1, ..., K$ of a recurrent neural network with K units. The edge between s_i and s_j is associated to a weight w_{ij}. The state of the network after each update is given by

$$\mathbf{s}^{t+1} = \varphi(\mathbf{W}\mathbf{s}^t),$$

where the $\mathbb{R}^{(N+1)\times(N+1)}$ matrix $\mathbf{W}$ contains the weights and φ is the nonlinear activation function.

A recurrent neural network can be 'unfolded' to a feed-forward structure with T layers by interpreting every time step as a separate layer, and thus be trained by *backpropagation through time*. This method has some unwanted properties, such as the exploding or vanishing gradient

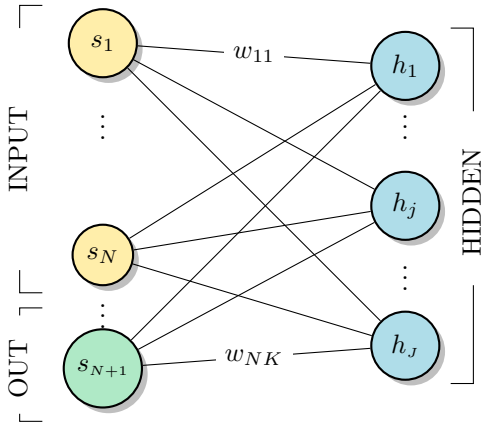


Figure 2.17: Graphical representation of a restricted Boltzmann machine with visible units $s_1, ..., s_{N+1}$ and hidden units $h_1, ..., h_J$, in which the visible units include input and output units. A full Boltzmann Machine would also allow for connections in between hidden units as well as in between visible units.

problem [41], and recurrent neural networks gained relatively little interest from the machine learning community in the last decades. However, and possibly due to the resemblance of the time-unfolded structure to deep neural networks (with the only difference that the weights are shared between the layers), this is currently changing [42]. Proposals to improve on training range from artificially 'clipping' the exploding gradients [41] to introducing time-delays between units [43].

An important class of recurrent neural networks for a task called pattern matching or associative memory are *Hopfield neural networks* [36]. Using binary units, symmetric all-to-all connections with $w_{ii} = 0$ for $i = 1, ..., N$, as well as the threshold activation function, one can easily show that for a given training point $s_1^m, ..., s_K^m$, choosing the weights w_{ij} proportional to $\langle s_i^{(m)} s_j^{(m)} \rangle, i, j = 1, ..., K$ leads to $\mathbf{s}^{(m)}$ being a stable state or 'attractor' of the network [16]. This means that an update of this state acts as the identity. In fact, the dynamics can be described as encoding the training patterns into an Ising-type energy function,

$$E(\mathbf{s}; \mathbf{W}) = -\mathbf{s}^T \mathbf{W} \mathbf{s} = -\frac{1}{2} \sum_{i,j=1}^{K} w_{ij} s_i s_j,$$

and the updating dynamics can be shown to drive the system state from the initial configuration to the closest energy minimum [44]. Hopfield networks are popular candidates for quantum computing applications due to their analogy with physical spin glass models.

2.3.2.4 Boltzmann machines

Boltzmann machines are probabilistic models which can be derived from recurrent neural networks but do not evolve in time. The K units are binary random variables divided into *visible units* $s = \{s_1, ..., s_{N+1}\}$ and *hidden units* $h = \{h_1, ..., h_J\}$ with $s_i, h_j \in \{0, 1\}$ for $i = 1, ..., N + 1$ and $j = 1, ..., J$. The visible units can again be divided into input and output units for supervised pattern recognition tasks, and for the binary classification problem considered here we assume that the first N visible units encode the inputs $x_1, ..., x_N$, and the last visible unit s_{N+1} encodes the output y. In more general, Boltzmann machines do not necessarily require the distinction

between inputs and outputs and can therefore be used for unsupervised learning. It is therefore convenient to summarise them in the set s. Note that one could of course write the random variables corresponding to the units in terms of vectors as before, but I will follow the more common approach and denote them as sets of variables.

A Boltzmann machine assigns the following parametrised probability distribution to possible values (or 'states') of s, h:

$$p(s,h;\theta) = \frac{1}{Z}\,\mathrm{e}^{-E(s,h;\theta)} \tag{2.21}$$

with the *partition function* summing over all (s,h)

$$Z = \sum_{s,h} \mathrm{e}^{-E(s,h;\theta)}, \tag{2.22}$$

and an Ising-type energy function

$$E(s,h;\theta) = -\sum_{ij} w_{ij}\; s_i h_j - \sum_{jj'} w_{jj'}\; h_j h_{j'} - \sum_{ii'} w_{ii'}\; s_i s_{i'}.$$

Boltzmann machines can be understood as Hopfield neural networks with stochastic binary units $s_i,\ i = 1...N+1$ whose probability to be in state 0 or 1 is given by

$$p(s_i = \pm 1) = \frac{1}{1 + e^{\mp 2\beta \sum\limits_j w_{ij} s_j}}. \tag{2.23}$$

The joint probability distribution is thus given by a Boltzmann distribution. The parameters θ of the model are given by the weights $\{w_{ij}, w_{jj'}, w_{ii'}\}$. The parameter β is a free model parameter corresponding to the temperature in statistical physics and in this context usually set to 1 [45]. In the graphical representation of neural networks, the random variables $s_1, ..., s_{N+1}$ are visible units while the $h_1, ..., h_J$ are hidden units. The parameters $w_{ij}, w_{jj'}$ and $w_{ii'}$ can be understood as weights connecting the hidden and visible units, the visible units amongst themselves and the hidden units amongst themselves, respectively. This is nothing else than a spin-glass model in statistical physics, where physical spins interact with different interaction strengths.

Since only the visible state s is of interest, one can marginalise over the hidden units by summing over all their configurations,

$$p(s;\theta) = \frac{1}{Z}\sum_h\; e^{-E(s,h)}. \tag{2.24}$$

which is the probability distribution $p(s;\theta)$ of a generative probabilistic machine learning model. Training a Boltzmann machine means to derive a parameter set $\theta = \{w_{ij}, w_{ii'}, w_{jj'}\}$ which gives rise to a model distribution from which the training dataset is a very likely sample.

It turns out that general Boltzmann machines are hard to train and the model only became popular when an efficient training algorithm for *restricted Boltzmann machines* was found [46] (see Figure 2.17). In a restricted Boltzmann machine visible units are only connected to hidden units and vice

versa, so that $w_{ii'} = w_{jj'} = 0$ and

$$E_{\mathrm{RBM}}(s,h) = -\sum_{ij} w_{ij}\, s_i h_j.$$

The basis of this training algorithm called *contrastive convergence* is based on ordinary gradient descent. The objective function is chosen according to maximum log-likelihood optimisation,

$$o(\theta) = \sum_{m=1}^{M} \log\ p(s^{(m)};\theta),$$

where $s^{(m)}$ represents the mth training pair. Inserting the formula for the probabilities from Eq. (2.24), the gradient used in the gradient descent update becomes

$$\begin{aligned}
\frac{\partial o(\theta,\mathcal{D})}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_{m=1}^{M} log\ p(s^m) \\
&= \sum_m \left(\frac{\partial}{\partial w_{ij}} log\ \sum_h e^{-E(s^m,h)} \right) - \frac{\partial}{\partial w_{ij}} log\ Z \\
&= \sum_m \frac{1}{\sum_h e^{-E(s^m,h)}} \frac{\partial \sum_h e^{-E(s^m,h)}}{\partial w_{ij}} - \frac{1}{Z} \frac{\partial \sum_{s,h} \mathrm{e}^{-E(s,h)}}{\partial w_{ij}} \\
&= \sum_m \sum_h \frac{e^{-E(s^m,h)}}{Z_{\mathcal{D}}} s_i^{(m)} h_j - \sum_{s,h} \frac{\mathrm{e}^{-E(s,h)}}{Z} s_i h_j,
\end{aligned}$$

where $Z_{\mathcal{D}} := \sum_{h,m} e^{-E(s^m,h)}$ is the partition function over the data set only, while Z includes a sum over all possible patterns s. This result is not surprising: The first term is the expectation value of the correlation between hidden and visible units over the data set, while the second term is the expectation value over all possible data produced by the model, and the goal of learning is to find a model where the two are as close as possible. The above expression is usually abbreviated as

$$\frac{\partial o(\theta,\mathcal{D})}{\partial w_{ij}} = \langle s_i h_j \rangle_{\mathrm{data}} - \langle s_i h_j \rangle_{\mathrm{model}}\,. \tag{2.25}$$

Calculating $\langle s_i h_j \rangle_{\mathrm{data}}$ is relatively simple [47], but even getting samples of $\langle s_i h_j \rangle_{\mathrm{model}}$ is intractable due to the partition function Z that involves computing a number of states which grows exponentially with N. A straight forward approach would be to approximate it with Gibbs sampling, which is a Markov Chain Monte Carlo method in which starting with an initial state, the values of the random variables $s_1, ..., s_{N+1}, h_1, ..., h_J$ are iteratively updated by drawing samples from the probability distribution 2.23. After a while the process 'thermalises' and values of (s,h) (with a sufficient amount of updates between them to avoid correlations) can be taken as samples for the Boltzmann distribution $\langle s_i h_j \rangle_{\mathrm{model}}$. However, thermalisation can be very slow and there is no method that indicates without fail whether an equilibrium is reached [48]. Also mean-field approximations known from statistical physics perform in most cases rather poor [14, p.989]. This was why Boltzmann machines were replaced by neural networks with backpropagation training algorithms in the 1980s [49], until in 2002 *contrastive divergence* was proposed [46] as a rather rough but effective approximation method. A number of quantum machine learning algorithms refer to this training method, which is why it shall be sketched briefly.

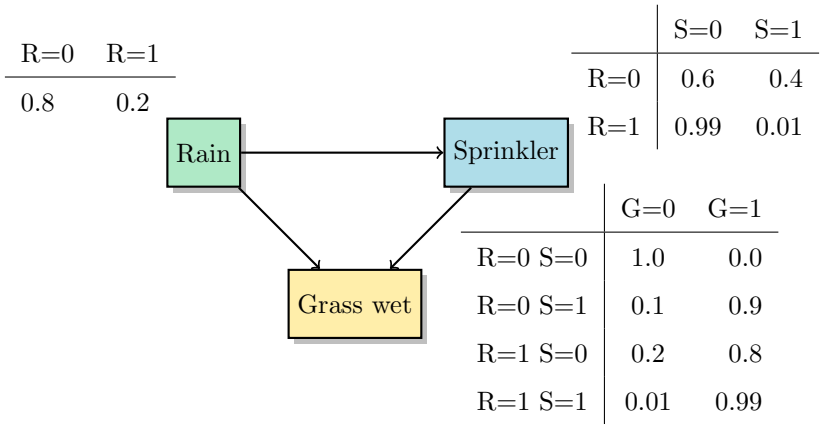


Figure 2.18: An example of a belief network with local conditional probability tables indicating the conditional probabilities of a child node given the state of its parents. The variable 'Rain' is conditional independent from the other two variables, while the variable 'Sprinkler' is only conditionally independent from the variable 'Grass wet'. If one knows the value of 'Rain' (the input) and one wants to derive the probability of 'Grass wet' (the output), the 'Sprinkler' becomes a hidden variable over which one has to marginalise.

Contrastive divergence is surprisingly simple: In each step the "positive gradient" is calculated by setting the visible units to a randomly picked training pair s^m. The $p(h_i)$ are calculated and values of h are drawn as a sample. Then the "negative gradient" is computed by calculating s' from the sampled h, after which one resamples h'. The weight update in Equation (2.25) is replaced by $sh - s'h'$. Against all intuition, only one step of this Gibbs sampling and resampling procedure is sufficient because the weights are updated in many iterations which overall induces an average of sorts[14] [49]. Although the contrastive divergence procedure actually does not lead to an update of the parameters according to the gradient of an existing objective function [50], it works well enough in many applications and became important for the training of deep (i.e., multi-layer) neural networks.

2.3.3 Graphical models

Graphical models are probabilistic models that use graphical representations to display and simplify probability distributions [15]. Again, $s = \{s_1, ..., s_{N+1}\}$ denotes a set of random variables representing the inputs $x_1 = s_1, ..., x_N = s_N$ and output $y = s_{N+1}$ of a machine learning task. They are assumed to take binary values so that the input and output domains become $\mathcal{X} = \{0,1\}^N, \mathcal{Y} = \{0,1\}$.

2.3.3.1 Bayesian Networks

A Bayesian network or belief network is a probabilistic model with conditional independence assumptions that simplify the model distribution $p(s)$. In probability theory, a general joint proba-

[14] The idea for this approach originated from the attempt to approximate an altogether different objective function, the difference between two Kullback-Leibler divergences [48]. The KL divergence between two distributions $a(x)$ and $b(x)$ is defined as $KL(a(x)||b(x)) = \sum_x a(x)\frac{a(x)}{b(x)}$. After T steps we get $\mathrm{CD}_t = \mathrm{KL}(p_0(s,h)||p_\infty(s,h)) - \mathrm{KL}(p_T(s,h)||p_\infty(s,h))$ in which p_T is the probability distribution after T steps of Gibbs sampling, beginning with an initial training data vector (so that p_0 is the distribution over the training data). The KL divergence can be interpreted as a free energy [49].

bility distribution over s can be expressed by the chain rule

$$p(s_1, ..., s_{N+1}) = p(s_1)p(s_2|s_1)p(s_3|s_1, s_2)...p(s_{N+1}|s_1, ..., s_N), \tag{2.26}$$

which follows directly from the definition of a conditional probability

$$p(s_{N+1}|s_N, ..., s_1) = \frac{p(s_{N+1}, ..., s_1)}{p(s_N, ..., s_1)}.$$

Two random variables a, b are *conditionally independent* given another random variable z if $p(a, b|z) = p(a|z)p(b|z)$. Assuming externally given conditional independences between the variables s_i reduces the conditional probabilities in Eq. (2.26) to $p(s_i|\pi_i)$, where π_i is the set of variables that s_i conditionally depends on. This reduces the original probability distribution to

$$p(s_1, ..., s_{N+1}) = \prod_{i=1}^{N+1} p(s_i|\pi_i). \tag{2.27}$$

For example, the factor $p(s_3|s_1, s_2)$ in Equation 2.26 reduces to $p(s_3|s_2)$ if s_3 is conditionally independent of s_1.

To use the model for prediction, the conditional probabilities $p(s_i|\pi_i)$ have to be derived from the data with methods discussed before. If they are parametrised by parameters θ_i, learning means to find the optimal parameters given the data, for example with maximum (log)-likelihood estimation,

$$\max_\theta \sum_{i,m} \log p(s_i^{(m)}|\pi_i, \theta_i). \tag{2.28}$$

Here, $s^m = (s_1^m, ..., s_{N+1}^m)$ is the m'th training point. To use Bayesian nets for prediction in the supervised learning case, one conditions $p(s)$ on the given input $s_1 = x_1, ..., s_N = x_N$ to get the class-conditional probability distribution $p(y|x_1...x_N)$ over the corresponding output variable $s_{N+1} = y$.

The important graphical representation of Bayesian nets as a directed acyclic graph makes these independence relations a lot clearer (see Figure 2.18). Each random variable corresponds to a node in the graph. The *parents* of a node are all nodes with a directed connection to it. The *nondescendants* of a node s_i are all nodes that cannot be reached by following the connections starting from s_i. The connectivity of a graph representing a Bayesian net follows the *Markov condition*: Any node is conditionally independent of its nondescendants given its parents. The parents of a node s_i therefore correspond to the variables π_i. The conditional probabilities $p(s_i|\pi_i)$ are "attached" to each node of the graph, for example as *local conditional probability tables*.

Note that the graph architecture can be understood as a hyperparameter similar to the choice of the number and size of layers in neural networks. Not only the local probabilities can be learnt, but also this structure of the graph. Structure learning is very difficult, and even with an infinitely large dataset one can only learn directed connections up to a property called Markov equivalence [51]. This stems from the fact that different directed graphs encode

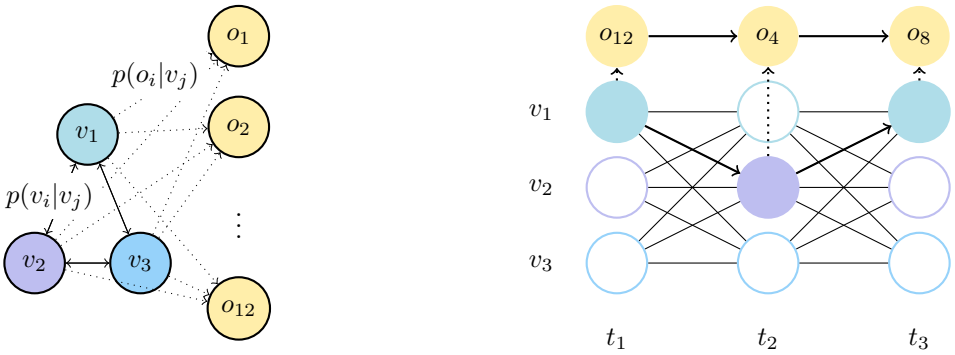


Figure 2.19: Illustration of a Hidden Markov model with three possible states v_1, v_2, v_3 and a set of possible observations $o_1, ..., o_{12}$. The transition probabilities $p(v_i|v_j), i, j = 1, 2, 3$ between the states as well as the probabilities $p(o_k|v_i)$ with $i = 1, 2, 3$ and $k = 1, ..., 12$ define the model and are illustrated by a graph on the left. A possible trajectory of the doubly stochastic Markov process unfolded in three time steps is sketched on the right: While the state jumps from v_1 to v_2 and back to v_1, the observer 'receives' observations o_{12}, o_4 and o_8.

the same conditional (in)dependence statements. Most algorithms define a scoring metric that measures the quality of a structure given the data and find better graphs by brute force search [52].

Also inference in Bayesian nets is generally a hard problem. In typical applications of Bayesian nets for inference, one observes values for some of the variables while others remain unknown. In other words, some of the variables of the Bayesian net are hidden variables whose value is not accessible. In the example in Figure 2.18 one might want to know the probability for the grass to be wet (output) given that it rained (input). Mathematically speaking, this means that the remainder of the s_i (the sprinkler) are hidden units h over which one has to marginalise, so that

$$p(y|x_1...x_N) = \sum_h p(y|x_1...x_N; h_1...h_K)$$

For binary units, the sum over h grows exponentially with K and inference becomes a NP-hard problem [53]. Some efficient inference algorithms for restricted classes of Bayesian nets are known, such as the *message passing algorithm* which is in $\mathcal{O}(K)$ (see [54] and references therein). Other solutions are approximate inference methods such as Monte Carlo sampling, in most cases with unknown accuracy and running time [14, Chapter 20.5].

2.3.3.2 Hidden Markov models

Hidden Markov models are graphical models whose properties are a little bit different from the preceding methods. Consider the problem of having a speech recording (i.e., a sequence of frequencies), and the task is to find the most likely sequence of words that correspond to the audio signal. The sequence of words can be modeled as a sequence of values for a random variable 'word', while the audio signal is a sequence of frequencies that can be represented by another random variable 'signal'. The input and output of the hidden Markov model are therefore sequences of values of random variables.

Given a random variable $v^{(t)}$ that can have different values $v_1, ..., v_L$, as well as (for now,

time-independent) transition probabilities $p(v_i|v_j)$ that indicate how likely it is that the state of v changes from value v_j to v_i. A (first order) *Markov process* updates the probability distribution $p(v^t) \to p(v^{t+1})$ according to these transition probabilities over time so that the current state only depends on the previous one. A possible sequence of a Markov process from time 0 to T shall be denoted as $V(T) = v^{(0)}, ..., v^{(T)}$.

To make it even more complicated, in hidden Markov models the states v of the system are unknown at any time (the corresponding units are hidden, and hence the name). The only known values are the 'observations' at time t modelled by a second random variable $o^{(t)}$ with possible values $\{o_1, ..., o_K\}$ [55]. What is also known are the probabilities $p(o_k|v_l)$ of an observation o_k being made given that the system was in state v_l. Sequences of observations up to time T are denoted by $O(T) = o^{(1)}, ..., o^{(T)}$. Hidden Markov models are therefore 'doubly embedded' stochastic processes. An example for a trajectory is illustrated in Figure 2.19 on the right, while the left sketches a graph for the two different kinds of transition probabilities.

The motivation behind this are machine learning tasks in which we have data which is a signature or hint of the actual information that we are interested in. In the speech recognition example, the states v may be the words uttered while the observation is the signal in the recording of a word. Given a recording of a speech as data, we are actually interested in the word sequence. The word sequence itself is modelled by a Markov process using transition probabilities that define how likely it is to have one word following another. The hidden Markov model can then be employed to find the sequence of words that is the most likely given the recording of a speech. Hidden Markov models also play an important role in many other applications such as text prediction, DNA analysis and online handwriting recognition [12].

To summarise the machine learning task, we are interested in is the most likely state sequence $\tilde{V}$ given an observation $\tilde{O}$, which is a typical supervised pattern recognition problem (called *state estimation* in this context [14]). The probabilistic model distribution of a hidden Markov model is given by

$$p(V(T), O(T)) = p(v^0) \prod_{t=2}^{T} p(v^t|v^{t-1}) \prod_{t=1}^{T} p(o^t|v^t).$$

In words, to find the probability of a sequence of states $V(T)$ and a sequence of observations $O(T)$ to occur together, one has to calculate the product of transitions between the states in the sequence, $p(v^0)p(v^1|v^0)....p(v^T|v^{T-1})$ multiplied by the product of probabilities of the observations made given the state, $p(o^0|v^0)p(o^1|v^1)...p(o^T|v^T)$. Learning in this context means to infer the transition probabilities $\{p(v_i|v_j), p(o_k|v_i)\}$ from a training data set.

Hidden Markov models are very close to the formalism of open quantum systems and have been generalised to 'hidden quantum Markov models' as discussed below.

2.3.4 Kernel methods

The following are some important nonparametric methods that make use of the concept of kernels.

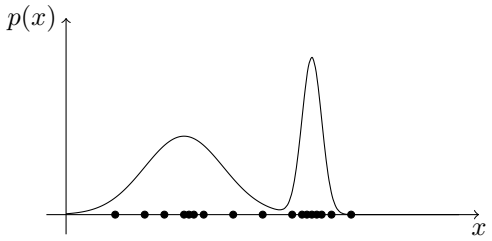


Figure 2.20: A kernel density estimator defines a probability distribution over the inputs (depending on the task, together with the outputs) that assigns high probabilities to regions with lots of data. The kernel function κ defines the shape of the resulting distribution.

2.3.4.1 Kernel density estimation

Kernel density estimation for pattern classification derives a class-conditional model distribution from data based on a simple idea. Given a similarity measure or kernel on the input space, define the class-conditional distribution for class $y = c$ as

$$p(x|y=c) = \sum_{m|y^m=c}^{M} \frac{1}{M_c} \kappa(\mathbf{x} - \mathbf{x}^{(m)}). \tag{2.29}$$

This is the sum of a distance measure κ between all M_c 'class c' training inputs. The total sum is larger if the training inputs from this class are close to the new input. This classifier is therefore based on the notion of 'similar inputs have similar outputs'. Also called a *Parzen window estimator*, the method is a smooth version of the histogram distribution in Figure 2.11 (see also Figure 2.20). Remember that the class-conditional probability is related to the desired distribution for prediction, $p(y|x)$, via Bayes formula in the Bayesian learning framework.

2.3.4.2 K-nearest neighbour

The k-nearest neighbour method is one of the most simple nonparametric methods. Given the dataset, one selects the k closest training inputs relative to the new input $\tilde{\mathbf{x}}$ according to a predefined distance metric on the input space (usually the Euclidean distance or Hamming distance). The predicted class label $\tilde{y}$ can be chosen according to the majority class amongst the neighbours for classification (see Figure 2.21), or as the average of their target outputs for regression. Variations to this simple algorithm include weighting the neighbours according to their distance [56], or replacing the training inputs of each class by their centroids. The k nearest neighbour method can be understood as a kernel density estimator with a uniform kernel and variable bandwidth hyperparameter h (which depends on the k nearest neighbours) [12]. Several authors have proposed quantum versions, and I will discuss two of them in Chapter 9.

2.3.4.3 Support vector machines

Support vector machines have been very popular throughout the 1990s, when they took over from neural networks as the method of choice. They can be derived from linear models for which one tries to minimise the distance between the separating hyperplane and the training vectors (see Figure 2.22). In other words, the model function is very simple, but the optimisation problem is more involved than the least-squares objective function. The power of support vector machines

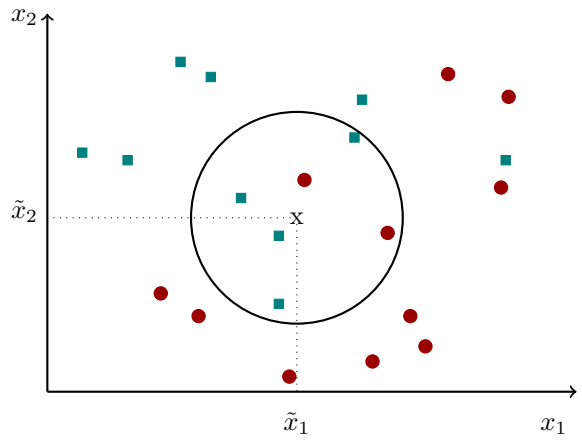


Figure 2.21: Illustration of k-nearest neighbour using the Euclidean distance measure. The symbols show the 2-dimensional inputs that have each a class attribute 'circle' or 'rectangle'. A new input $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x_2})$ is classified according to its $k = 6$ nearest neighbours (i.e., taking the class label of the majority of its neighbours, in this case 'rectangle').

only appears when the inner product kernels appearing in the objective function are replaced by other kernels that effectively implement feature maps [57].

The following Box outlines the somewhat technical derivation of the model following the description of [12] and [18]. I consider the binary classification problem $\mathcal{X} = \mathbb{R}^N, \mathcal{Y} = \{-1, 1\}$.

Box 2.3.1: Derivation of the support vector machine model

Start with a linear model,

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0, \tag{2.30}$$

The idea is to find a set of parameters $\mathbf{w}$ which defines a hyperplane that separates the training data such that points of the two classes are on either side of the hyperplane, in addition to a certain margin. Mathematically this can be expressed by $f(\mathbf{x}^m; \mathbf{w}) \geq 1$ for data points with $y^m = 1$, and $f(\mathbf{x}^m; \mathbf{w}) \leq 1$ for data points of class $y^m = -1$. In summary, one can write

$$f(\mathbf{x}^m; \mathbf{w}) y^m \geq 1 \tag{2.31}$$

for $m = 1, .., M$. (Note that if the data is not separable, one can introduce slack variables ξ^m that change the right side of the inequality to $1 - \xi^m$ and minimise the slack variables along with the other parameters below, introducing another tuning parameter γ.) The goal is to find the decision boundary that maximises the *margin* or distance

$$d^s = \frac{|\mathbf{w}^T \mathbf{x}^s + w_0|}{|\mathbf{w}|}$$

between the closest training points (the *support vectors* $\mathbf{x}^s$) and the separating hyperplane, while ensuring condition (2.31) is fulfilled. Fixing one degree of freedom, the length of the

weight vector, with $d|\mathbf{w}| = 1$ the optimisation problem becomes equivalent to maximising $|\mathbf{w}|^{-1}$ (alternatively , one usually minimises $\frac{1}{2}|\mathbf{w}|^2$) under the constraint $f(\mathbf{x}^m;\mathbf{w})y^m \geq 1$. Using Lagrangian multipliers $\alpha_1, ..., \alpha_M$, this constrained quadratic optimisation problem can be turned into an unconstrained version in which we need to minimise the Lagrange function

$$L(\mathbf{w}, \{\alpha\}, w_0) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{m=1}^{M} \alpha_m \big(y^m(\mathbf{w}^T\mathbf{x}^m + w_0) - 1\big).$$

The last step is to formulate the dual problem which only depends on the size of the dataset M. Setting the derivative of L to zero leads to the relation $\mathbf{w} = \sum_{m=1}^{M} \alpha_m y^m \mathbf{x}^m$ as well as $\sum_m \alpha^m y^m = 0$, and resubstituting these results in the objective function yields the Lagrangian function

$$L_d(\{\alpha\}) = \sum_{m=1}^{M} \alpha_m - \frac{1}{2} \sum_{m,m'=1}^{M} \alpha_m \alpha_{m'} y^m y^{(m')} \mathbf{x}^m, \tag{2.32}$$

that has to be minimised with respect to the Lagrangian multipliers α_m and subject to the constraints $\alpha_m \geq 1\ \forall m$ and $\sum_m \alpha_m y^m = 0$.

It becomes apparent that support vector machines are nonparametric models, as the number of parameters $\{\alpha_1 ... \alpha_M\}$ grows with the training set size. Once the Lagrangian parameters are found, the weights are determined by the relation

$$\mathbf{w} = \sum_{m=1}^{M} \alpha_m y^m \mathbf{x}^m \tag{2.33}$$

and the new output is found as

$$\tilde{y} = (\sum_{m=1}^{M} \alpha_m y^m \mathbf{x}^m + w_0)\tilde{\mathbf{x}} \tag{2.34}$$

Note that Equation (2.32) defines the optimisation problem connected to a support vector machine. The scalar products $\mathbf{x}^m \mathbf{x}^{(m')}$ are a kernel to which the kernel trick can be applied. Solving this optimisation problem takes time in $\mathcal{O}(M^3)$. I will later mention a slightly different formulation of the problem that led to one of the first quantum machine learning algorithms.

2.3.4.4 Gaussian processes

Given a supervised regression task with $\mathcal{X} = \mathbb{R}^N, \mathcal{Y} = \mathbb{R}$. The idea behind the method of Gaussian processes is to assign a probability to every possible model function $f(\mathbf{x})$ favouring those we consider more likely, e.g. smooth functions. Out of the resulting 'probability distribution over functions' one selects only those that agree with the data points in $\mathcal{D}$ to get a 'posteriori over functions' in the Bayesian learning framework (see Box 2.2.3). The mean of this posteriori can be understood as the best guess for $f(\tilde{\mathbf{x}})$ at a certain point $\tilde{\mathbf{x}}$ given the data, and the variance is the uncertainty of this guess.

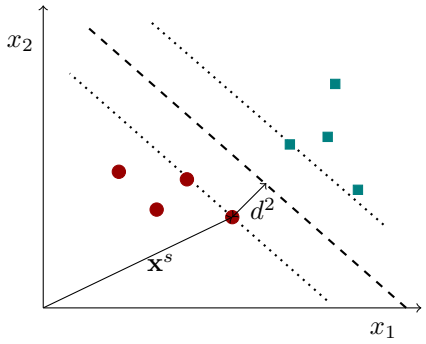


Figure 2.22: A support vector machine is derived from the problem of finding the discriminant hyperplane with the largest margin d^s to the input vectors.

The details are again based on various technicalities. Gaussian processes are "collection of random variables, any finite number of which have a joint Gaussian distribution" [30]. In this context the random variables are the outputs of the model function $f(\mathbf{x})$, and a Gaussian process can be understood as a generalisation of Gaussian probability distributions to functions, formally written as

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}')).$$

The process is fully defined by the mean $m(\mathbf{x})$ of all functions at each point $\mathbf{x}$ (which is usually, and in the following, set to zero), as well as the covariance between two points, $\kappa(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')]$. Given such a covariance function $\kappa(\mathbf{x}, \mathbf{x}')$ and a number of inputs $\mathbf{x}_1, ..., \mathbf{x}_N$, the according outputs $f(\mathbf{x}_1), ..., f(\mathbf{x}_n)$ are random variables sampled from the joint distribution

$$\begin{pmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_N) \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} \right]. \tag{2.35}$$

The covariance function is responsible for the type of distribution over functions defined by a Gaussian process. Training a Gaussian Processes refers to finding suitable covariance functions. Common choices are the dot product

$$\text{cov}(f(\mathbf{x}_1, f(\mathbf{x}_2))) = \kappa(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{x}_2,$$

or the squared exponential

$$\text{cov}(f(\mathbf{x}_1, f(\mathbf{x}_2))) = \kappa(\mathbf{x}_1, \mathbf{x}_2) = \sigma_f^2 e^{-\frac{1}{2l^2}|\mathbf{x}_1 - \mathbf{x}_2|^2}, \tag{2.36}$$

where σ_f^2, l are hyperparameters. The covariance matrix made of covariance functions from a set of inputs is positive semi-definite and therefore a kernel Gram matrix, which is why I denoted the entries with κ.

Equation (2.35) shows how to draw samples of values of a function $f(\mathbf{x})$ at a number of points (and if we increase the number of inputs we can get a good idea of how the function consequently sampled from the Gaussian process looks like), and we can use it for prediction. Instead of arbitrary

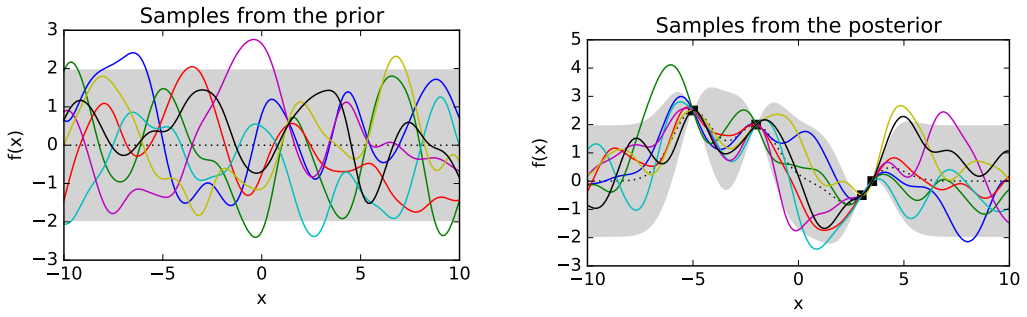


Figure 2.23: Prior (left) functions drawn from a Gaussian process. The covariance determines the smoothness of the sample functions. Data reduces the possible samples and and introduces certainty where the data values are. Samples from the posterior (right) are therefore reduced to model functions that agree with the data points. *The plots were made with the python infby package written by John Reid. Accessed through http://pythonhosted.org/infpy/.*

inputs, we want to get the joint distribution for the training data outputs as well as the new output $f(\tilde{\mathbf{x}})$,

$$\begin{pmatrix} f(\mathbf{x}^1) \\ \vdots \\ f(\mathbf{x}^M) \\ f(\tilde{\mathbf{x}}) \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \dots & \kappa(\mathbf{x}^1, \mathbf{x}^M) & \kappa(\mathbf{x}^1, \tilde{\mathbf{x}}) \\ \vdots & \ddots & \vdots & \vdots \\ \kappa(\mathbf{x}^M, \mathbf{x}^1) & \dots & \kappa(\mathbf{x}^M, \mathbf{x}^M) & \kappa(\mathbf{x}^M, \tilde{\mathbf{x}}) \\ \kappa(\tilde{\mathbf{x}}, \mathbf{x}^1) & \dots & \kappa(\tilde{\mathbf{x}}, \mathbf{x}^M) & \kappa(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}) \end{pmatrix} \right]. \tag{2.37}$$

The desired probability distribution $p(\tilde{y}|\tilde{\mathbf{x}}, \mathcal{D})$ over the predicted output $f(\tilde{\mathbf{x}}) = \tilde{y}$ is now a marginalisation of Eq. (2.37) for the fixed values $f(\mathbf{x}^1) = y^1, ..., f(\mathbf{x}^M) = y^M$ which can be thought of as 'cutting' through the joint probability distribution in order to reduce the multivariate Gaussian to a univariate (one-dimensional) Gaussian. The technical algebraic details of obtaining a marginal Gaussian distribution can be found in most statistics textbooks and I only write down the result here[15]:

$$p(y|\mathbf{x}, \mathcal{D}) = \mathcal{N}\big[y| \underbrace{\boldsymbol{\kappa}^T \mathbf{K}^{-1} \mathbf{y}}_{\text{mean } \mu}, \underbrace{\kappa - \boldsymbol{\kappa}^T \mathbf{K}^{-1} \boldsymbol{\kappa}}_{\text{covariance } \delta}\big], \tag{2.38}$$

with the notation

$$\boldsymbol{\kappa} = \begin{pmatrix} \kappa(\mathbf{x}^{(1)}, \tilde{\mathbf{x}}) \\ \vdots \\ \kappa(\mathbf{x}^{(M)}, \tilde{\mathbf{x}}) \end{pmatrix}, \mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \dots & \kappa(\mathbf{x}^{(1)}, \mathbf{x}^{(M)}) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^{(M)}, \mathbf{x}^{(1)}) & \dots & \kappa(\mathbf{x}^{(M)}, \mathbf{x}^{(M)}) \end{pmatrix}, \kappa = \kappa(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}), \mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(M)} \end{pmatrix}.$$

Gaussian processes do not have a distinct learning phase. With Equation (2.38), prediction is mainly a computational problem of efficiently inverting the $M \times M$-dimensional kernel matrix $\mathbf{K}$. Gaussian processes do take a lot of computational resources when large datasets have to be processed ($M > 10,000$) since the matrix inversion scales with $\mathcal{O}(M^3)$ [14], and many proposals for

[15]Note that I only consider noise free data here, however, including Gaussian noise simply adds a variance parameter to the matrix $\mathbf{K}$ (for more details, see [30]).

approximations have been put forward [30]. A large share of the effort in Gaussian processes also goes into finding a suitable kernel or covariance function. Since these are in general parametrised with hyperparameters θ (as in Eq. (2.36)) such as the correlation length or the precision of the noise, we still end up with a parameter optimisation problem of finding the parameters that maximise $p(y^1, ..., y^M|\theta)$.

Chapter 3

Quantum computing

While the last chapter introduced readers without a firm background in machine learning into the foundations necessary for the following parts of the thesis, this chapter gives an overview of some concepts and special methods required from the field of quantum computing. I assume the reader is familiar with quantum theory. The mathematical formalism is briefly summarised in the next section in order to introduce notation and terminology.

3.1 Synopsis of quantum theory

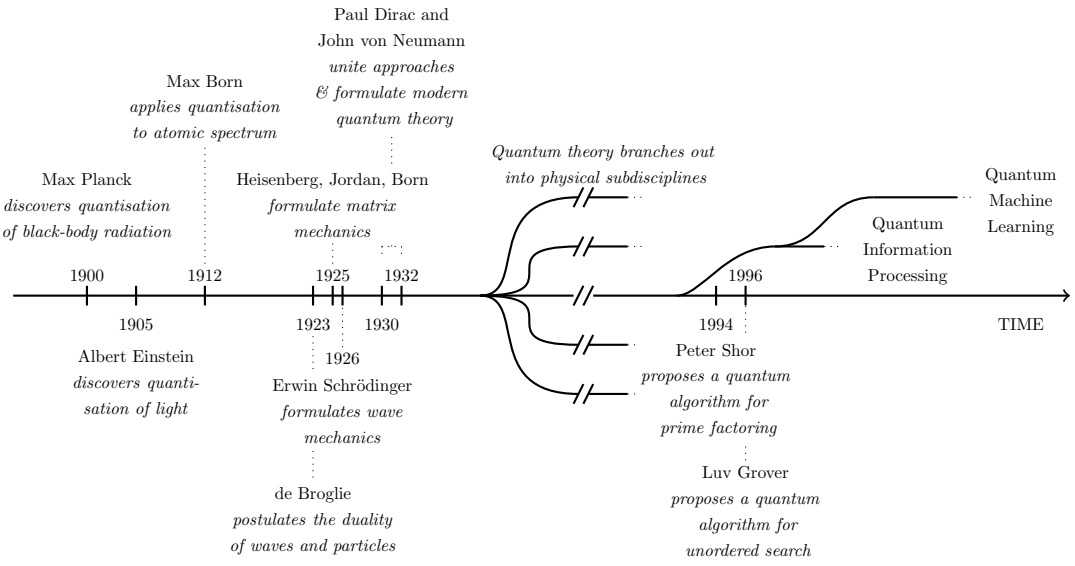
Quantum theory is understood here to be "first and foremost a calculus for computing the probabilities of outcomes of measurements made on physical systems" [58] that we call quantum systems. Scott Aaronson, in his online lecture notes[1] that led to the book "Quantum Computing since Democritus" [59], remarks sarcastically that in order to learn about quantum theory,

> [...] you start with classical mechanics and electrodynamics, solving lots of grueling differential equations at every step. Then you learn about the "blackbody paradox" and various strange experimental results, and the great crisis these things posed for physics. Next you learn a complicated patchwork of ideas that physicists invented between 1900 and 1926 to try to make the crisis go away. Then, if you're lucky, after years of study you finally get around to the central conceptual point: that nature is described not by probabilities (which are always non-negative), but by numbers called amplitudes that can be positive, negative, or even complex.

From the pure perspective of quantum computing this comment certainly contains some truth, as here only abstract and simple kinds of quantum systems called *qubits* measured by the most simple type of measurements, projective measurement in the computational basis, are of importance. **I will therefore also focus on discrete and finite-dimensional systems for the remainder of this chapter.** However, for more complicated models - in quantum information processing and elsewhere- quantum theory is a lot richer than replacing probabilities with complex numbers, as the extensive literature over nearly a century witnesses. A brief historical overview of quantum theory can be found in the followig Box 3.1.1.

[1] http://www.scottaaronson.com/democritus/ [October 2016].

Box 3.1.1: Highlights in the history of quantum theory



What Aaronson describes as a "complicated patchwork of ideas that physicists invented between 1900 and 1926" forms the beginnings of quantum theory as it is still taught today, and which was then used to rethink the entire body of physics knowledge. The initial step is commonly attributed to the year 1900 when Max Planck introduced the idea that energy (in this case so called *black-body radiation*) can only be absorbed in discrete portions as if existing as 'quanta' or small energetic portions. With this assumption he was able to resolve a heated debate regarding the spectrum of black-body radiation [60]. A few years later, Albert Einstein made a similar discovery with light emission in the photoelectric effect, and derived the concept of a photon - a portion or energy quantum of light [61, 62]. In the following years, these early ideas of a 'theory of energy quanta' were applied to atomic spectroscopy (most notably by Niels Bohr), and to light (by Luois de Broglie) but still based on rather ad-hoc methods [60]. Werner Heisenberg followed by Jordan, Born and, independently, Paul Dirac formulated the first mathematically consistent version of quantum theory referred to as matrix mechanics in 1925, with which Wolfgang Pauli was able to derive the experimental results of measuring the spectrum of a hydrogen atom. Heisenberg postulated his uncertainty principle shortly after, stating that certain properties of a quantum system cannot be measured accurately at the same time. In 1926, following a slightly different and less abstract route, Erwin Schrödinger proposed his famous equation of motion for the 'wave function' describing the state of a quantum system. These two approaches were shown to be equivalent in the 1930s, and a more general version was finally proposed by Dirac and Jordan. In the following years, quantum theory branched out into many subdisciplines and many traditional pillars of physics were reformulated in this new framework.

The mathematical apparatus to calculate the probabilities and expectation values of measurement outcomes for a quantum system is largely based on linear algebra calculus. One associates the measurable properties of a quantum system with a vector $|\psi\rangle$ in a **Hilbert space** $\mathcal{H} = \mathbb{C}^N$ called

a **quantum state**. The scalar product defined on the Hilbert space is denoted as $\langle\psi|\psi\rangle$.

An **observable** is a physical property or variable and is represented by a Hermitian operator O on $\mathcal{H}$. It follows from the **spectral theorem** of linear algebra that *there exists a orthonormal basis of $\mathcal{H}$ consisting of eigenvectors of O*, and the corresponding eigenvalues are real. For a discrete and finite-dimensional system associated with a N-dimensional Hilbert space $\mathbb{C}^N$, every $|\psi\rangle \in \mathbb{C}^N$ can hence be expressed in O's eigenbasis $\{|\psi_i\rangle\}_{i=1...N}$,

$$|\psi\rangle = \sum_{i=1}^{N} a_i|\psi_i\rangle,$$

where the $a_i \in \mathbb{C}$ are the (quantum) amplitudes. The effect of applying O to an element $|\psi\rangle \in \mathbb{C}^N$ is fully defined by the N eigenvalue equations $O|\psi_i\rangle = \lambda_i|\psi_i\rangle$ with eigenvalues λ_i. **Expectation values** of the observable property are calculated by

$$\mathbb{E}(O) = \langle\psi|O|\psi\rangle = \sum_{ij} a_i^* a_j \langle\psi_i|O|\psi_j\rangle = \sum_{ij} a_i a_j \lambda_j \langle\psi_i|\psi_j\rangle = \sum_i |a_i|^2 \lambda_j$$

The dynamic evolution of a quantum state is represented by a **unitary operator** $U = U(t_2, t_1)$ mapping $|\psi(t_1)\rangle$ to $U(t_2, t_1)|\psi(t_1)\rangle = |\psi(t_2)\rangle$, where the property $U^\dagger U = 1$ (with $U^\dagger$ being the Hermitian conjugate) ensures that states remain normalised. U is the solution of the corresponding Schrödinger equation $i\hbar\partial_t|\psi\rangle = H|\psi\rangle$ with Hamiltonian H. For time-independent Hamiltonians one can write the solution as $U = e^{-iHt}$.

Quantum theory can also be formulated in terms of the outer product of a vector in $\mathcal{H}$, a **density operator** $\rho = |\psi\rangle\langle\psi|$, which is a trace-1 Hermitian positive operator. A density operator allows us to elegantly incorporate classical probability theory "on top" of quantum theory by considering an *ensemble* or *mixture* of quantum states, $\rho = \sum_k p_k|\psi_k\rangle\langle\psi_k|$ with $\sum_k p_k = 1$ and p_k being classical probabilities of the system being in state $|\psi_k\rangle$. The **expectation value** for a state described by a density operator is calculated by $\mathbb{E}(O) = \mathrm{tr}\{\rho O\}$. The most general dynamics (mixing 'coherent' quantum and 'incoherent' classical probability theory) is a **(completely) positive trace-preserving map** from a density operator to a density operator.

A **joint quantum system** is described by considering the tensor product of the Hilbert spaces of the subsystems, $\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B$. If $|\phi_i\rangle,\ i = 1...N_a$ is a basis of the discrete finite Hilbert space $\mathcal{H}_A$ and $|\varphi_j\rangle,\ j = 1...N_b$ a basis of $\mathcal{H}_B$, then a basis of the composite space is made up of the $N_a N_b$ basis states $|\phi_i\rangle \otimes |\varphi_j\rangle$. For the tensor products of quantum states the shorthand $|\phi_i\varphi_j\rangle$ is frequently used.

The **partial trace** is used to obtain the description of a subsystem from a composed system. In Dirac notation the trace over subsystem A of a joint quantum system $\mathcal{H}_B$ with basis $\{|\psi_{b_i}\rangle\}$ reads

$$\rho_A = \mathrm{tr}_B \rho_{AB} = \sum_{i=1}^{N} \langle\psi_{b_i}|\rho_{AB}|\psi_{b_i}\rangle.$$

For discrete and finite-dimensional systems, quantum states and observables can be elegantly expressed by complex vectors and matrices. The ith element in the **vector representation**

of $|\psi\rangle$ in the basis $\{|\psi_1\rangle, ..., |\psi_N\rangle\}$ can be calculated by $\langle\psi_i|\psi\rangle$. The entry ij of the **matrix representation** of observable O can be calculated by $\langle\psi_i|O|\psi_j\rangle$. The unitary dynamics can be expressed by a unitary matrix applied to the quantum state vector.

3.2 Quantum information processing

As quantum theory motivated physicists to rethink all aspects of their discipline from a new perspective, it was only a matter of time before the question arose what quantum mechanics means to information processing. In a way, with the debates of nonlocality and Einstein's "spooky action at a distance" this has been part of the early days of quantum mechanics. However, it took until the late 1980s for quantum information processing research to form an independent subdiscipline. Central questions are: *What is quantum information? Can we build a new type of computer based on quantum systems? How can we formulate algorithms on such machines? What does quantum theory mean for the limits of what is computable?*

In the media (and possibly much fuelled by researchers themselves hunting for grants), quantum computing is still portrayed as the cure for all, mirrored in the following remark from a machine learner's perspective:

> Much like artificial intelligence in its early days, the reputation of quantum computing has been tarnished by grand promises and few concrete results. Talk of quantum computers is often closely flanked by promises of polynomial time solutions to NP-Hard problems and other such implausible appeals to blind optimism. [63, p. 3]

It is true that after more than 20 years of establishing an independent research discipline, and not-surprisingly, there is still no final answer to many of the questions posed. Most prominently, we still do not know for sure whether $BQP \neq BPP$, or whether the class of decision problems for which a solution can be found with a constant probability > 0.5 using a quantum Turing machine is larger than when using a classical Turing machine. Nevertheless, there is a lot more we know. For example, we know that there are quantum algorithms that grow slower in runtime with the size of the input than known classical algorithms solving the same problem [64, 65, 66]. Relative to a black-box function or oracle, quantum algorithms are even proven to be faster than any *possible* classical algorithm. Another important result is that every classical algorithm can be implemented on a quantum computer with only polynomial overhead, so in theory a quantum computer is at least as good as a classical computer [67]. A rich landscape of quantum computational models and routines has been formulated [68], and a database of algorithms is frequently updated by Stephen Jordan at the American National Instititute of Standards and Technology.[2]

3.2.1 Quantum algorithms

Although there are a variety of computational models that formalise the idea of quantum information, all of them are more or less based on the concept of a *qubit*, which is a quantum system associated with only two measurable events or a two-dimensional Hilbert space. Many popular

[2] http://math.nist.gov/quantum/zoo/, last visited February 2017.

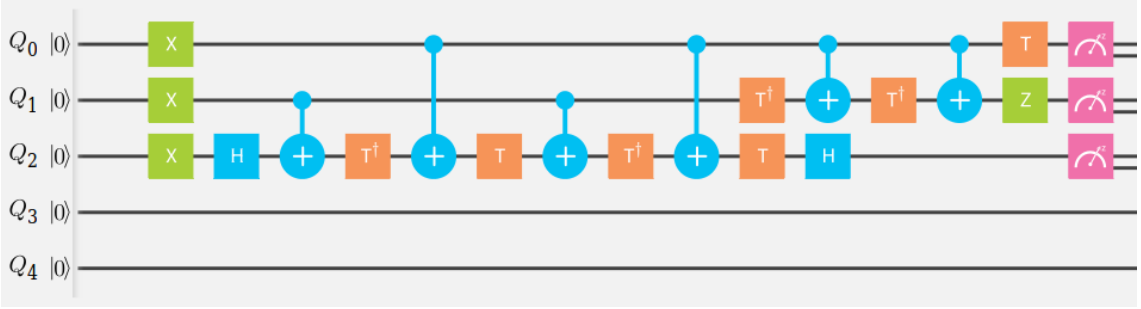


Figure 3.1: Quantum circuit of IBM's 5-qubit Quantum Experience' online interface [69]. The lines each represent one qubit. The squares represent simple single qubit gates such as the X, Hadamard, phase shift and Z gate. The circles denote a controlled 2 qubit operation. The pink boxes at the end indicate which qubits are measured.

notions of quantum information claim that the power of quantum computers stems from the fact that qubits can be in a linear combination of 0 and 1. But also a classical random bit (i.e. a classical coin toss) have this property to a certain extent. This is why sampling algorithms are often the most suitable competitors to quantum routines. Two major differences stemming from the mathematical formalism of quantum theory are:

1. The coefficients of the linear combination in the quantum case can be complex, which means they can be negative. This allows for interference effects when the state of a qubit is evolved by unitary evolutions

2. Observables in quantum theory can be non-commutative

A quantum computer can be understood as a physical implementation of n qubits with a precise control on the evolution of the state. A quantum algorithm is a targeted manipulation of the quantum system which can always be described by a unitary map from an initial state of the overall quantum system $|\psi\rangle$ to a desired final state $|\psi'\rangle$, with which an input-output relation is computed. An important theorem in quantum information states that any unitary can be approximated by a sequence of elementary unitaries which only act on the Hilbert spaces of one or two qubits [70]. Based on this insight, quantum algorithms are widely formulated as circuits of elementary unitaries or *quantum gates* (see Figure 3.1). A universal quantum computer consequently only has to know how to perform a small set of operations on qubits, just like classical computers are build on a limited number of logic gates. Runtime considerations usually count the number of quantum gates it takes to implement the entire quantum algorithm. *Efficient* quantum algorithms are based on unitaries whose decomposition grows at most polynomially with the input size of the problem.

As much as a classical bit is an abstract model of a binary system that can have many different physical realisations, a qubit can be used as a model of many different quantum systems. Some current candidates for implementations are superconducting qubits [71], photonic setups [72], ion traps [73] or topological properties of quasi-particles [74]. Each of them has advantages and disadvantages, and it is not unlikely that future architectures use a mixture of these implementations.

Amplitude	Index	basis state	shorthand
a_0	$000 \hat{=} 0$	$\|000\rangle$	$\|0\rangle$
a_1	$001 \hat{=} 1$	$\|001\rangle$	$\|1\rangle$
a_2	$010 \hat{=} 2$	$\|010\rangle$	$\|2\rangle$
a_3	$011 \hat{=} 3$	$\|011\rangle$	$\|3\rangle$
a_4	$100 \hat{=} 4$	$\|100\rangle$	$\|4\rangle$
a_5	$101 \hat{=} 5$	$\|101\rangle$	$\|5\rangle$
a_6	$110 \hat{=} 6$	$\|110\rangle$	$\|6\rangle$
a_7	$111 \hat{=} 7$	$\|111\rangle$	$\|7\rangle$

Table 3.1: The index of an amplitude from the 2^3-dimensional quantum state vector has a 3-bit binary representation that is exactly the basis state or event it is referring to.

3.2.2 Qubits and quantum gates

To recapitulate, a qubit is a quantum system with two possible measurement outcomes or possible events, and its state is an element of a two dimensional Hilbert space $\mathcal{H}_2$. The quantum state of n qubits is an object in $\mathcal{H}_{2^n} = \mathcal{H}_2 \otimes \ldots \otimes \mathcal{H}_2$. Let $\{|\psi_i\rangle\}$ be a basis of the joint Hilbert space, which consists of $2^n = N$ basis states. A quantum state of an n-qubit system can then be expressed as

$$|\psi\rangle = \sum_{i=0}^{N-1} a_i |\psi_i\rangle, \quad \text{with} \quad a_i \in \mathbb{C}, \quad \sum_{i=0}^{N-1} |a_i|^2 = 1.$$

In vector representation, the Hilbert space can be chosen as the $\mathbb{C}_{2^n} = \mathbb{C}_2 \otimes \ldots \otimes \mathbb{C}_2$ with basis vectors $\{\mathbf{a}_i\}$ and the quantum state becomes

$$\mathbf{a} = \sum_{i=0}^{N-1} a_i \mathbf{a}_i, \quad \text{with} \quad \mathbf{a}^\dagger \mathbf{a} = 1.$$

Unless otherwise stated, I will use the standard basis when using the vector formulation, which corresponds to the computational basis in Dirac notation,

$$|0...0\rangle \leftrightarrow \begin{pmatrix} 1 \\ \vdots \\ 0 \end{pmatrix}, \ldots, |1...1\rangle \leftrightarrow \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix},$$

so that a quantum state is given by

$$\mathbf{a} = \begin{pmatrix} a_0 \\ \vdots \\ a_{N-1} \end{pmatrix}$$

in matrix notation and by

$$\begin{aligned} |\psi\rangle &= a_0 |0\rangle \otimes ... \otimes |0\rangle + \ldots + a_{N-1} |1\rangle \otimes ... \otimes |1\rangle \\ &= a_0 |0...0\rangle + \ldots + a_{N-1} |1...1\rangle \end{aligned}$$

in Dirac notation. Running the index i from 0 to $N-1$ in the Dirac representation has the elegant

property that the amplitude at position i refers to the event of measuring the qubits in the n-bit binary representation of integer i (see Table 3.1). A general n-bit quantum state in Dirac notation can therefore be compactly written as

$$|\psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle. \tag{3.1}$$

The corresponding density matrix is given by

$$\rho_{\text{pure}} = |\psi\rangle\langle\psi| = \sum_{i,j=0}^{N-1} a_i^* a_j |i\rangle\langle j|. \tag{3.2}$$

For a general mixed state the coefficients do not factorise and we get

$$\rho_{\text{mixed}} = \sum_{i,j=0}^{N-1} a_{ij}|i\rangle\langle j|, \qquad a_{ij} \in \mathbb{C}. \tag{3.3}$$

As a general rule, if there is an index between the Dirac bracket it refers to the corresponding basis state in computational basis.

As mentioned above, quantum gates are elementary unitary operators or building blocks from which larger algorithms can be built. The action of those operators onto quantum states is fully defined by their action on each eigenvector of an eigenbasis. It is practical to choose the computational basis as an eigenbasis. For example, using the basis $\{|0\rangle, |1\rangle\}$ the single qubit Hadamard gate

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \;\hat{=}\; \frac{1}{\sqrt{2}}(|0\rangle\langle 0| + |1\rangle\langle 0| + |0\rangle\langle 1| - |1\rangle\langle 1|)$$

implements

$$\begin{aligned} H|0\rangle &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ H|1\rangle &\rightarrow \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

Another example is the single qubit gate σ_z

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \;\hat{=}\; |0\rangle\langle 0| - |1\rangle\langle 1|.$$

which writes a sign in front of qubits in the 1 state, while leaving qubits in the 0 state as they are. The eigenstates of σ_z are the computational basis states,

$$\sigma_z|0\rangle = |0\rangle,\ \sigma_z|1\rangle = -|1\rangle.$$

The textbook by Michael Nielsen and Isaac Chuang is still an excellent introduction into qubits and gates [68].

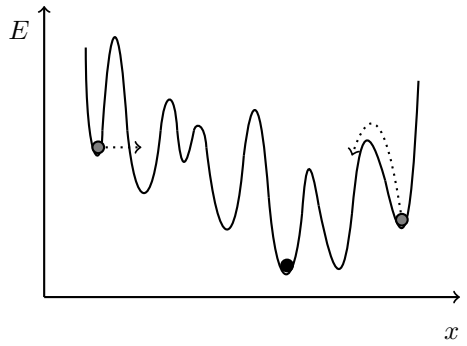


Figure 3.2: Illustration of quantum annealing in an energy landscape over (here continuous) states or configurations x. The ground state is the configuration of lowest energy (black dot). Quantum tunnelling allows the system state to transgress high and thin energy barriers (gray dot on the left), while in classical annealing techniques stochastic fluctuations have to be large enough to allow for jumps over peaks (gray dot on the right).

3.2.3 Quantum annealing and other computational models

Although the circuit model of qubits and gates is by far the most common formalism there are some other computational models. These have so far been shown to be equivalent up to a polynomial overhead, which means that efficient translations from one to the other are known.

Prominent in the quantum machine learning literature is a technique called *quantum annealing*, which can be understood as a heuristic to *adiabatic quantum computing*. Adiabatic quantum computing [75] is in a sense the analogue version of quantum computing [76] in which the solution of a computational problem is encoded in the ground state (i.e. lowest energy state) of a Hamiltonian which defines the dynamics of the system. Starting with a quantum system in the ground state of another Hamiltonian which is relatively simple to realise in a given experimental setup, and slowly adjusting the system so that it is governed by the desired Hamiltonian shall ensure that the ground state is found. Adjusting a Hamiltonian can be realised by changing field and interaction strengths between the physical objects that realise the qubits.

It turns out that for many problems, to keep the system in the ground state during the adjustment ('annealing schedule') requires a very slow evolution from one to the other Hamiltonian, and often a time exponential in the problem size, which shows once more that nature seems to set some universal bounds for computation. Quantum annealing may be seen as a heuristic or 'shortcut' to the adiabatic algorithm that works much like simulated annealing in computer science (see Boltzmann machine training in Section 2.3.2.4). The main difference between classical and quantum annealing is that thermal fluctuations are replaced by quantum fluctuations which enables the system to *tunnel* through high and thin energy barriers (the probability of quantum tunnelling decreases exponentially with the barrier width, but is independent of its height). That makes quantum annealing especially fit for problems with a sharply ragged objective function (see Figure 3.2).

The great interest in quantum annealing is driven by the relatively mature hardware implementation. Besides the hardware available in laboratories, a commercially available product, the D-Wave

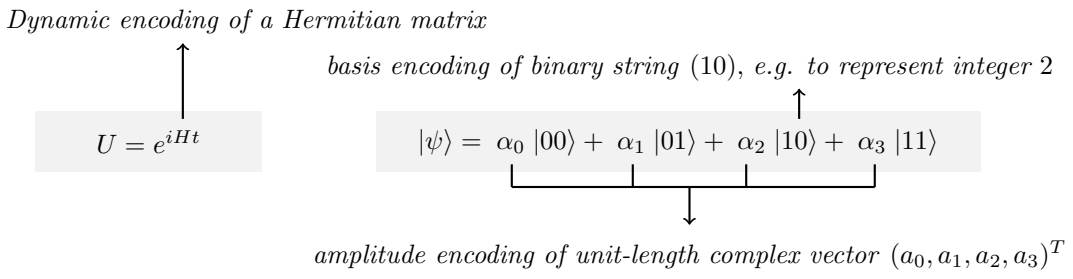


Figure 3.3: Illustration of the different encoding strategies for a quantum state of a 2-qubit system.

quantum annealer[3], has been tested since a number of years by large research collaborations and gives insights into the practical challenges of quantum computing (which still seem comfortably distant in the case of universal quantum computers). Current devices are limited to solving quadratic unconstrained binary optimisation (QUBO) problems,

$$\min_{(x_1,\ldots,x_N)} \sum_{ij} w_{ij}\, x_i x_j \quad \text{with} \quad x_i, x_j \in [0,1],\ w_{ij} \in \mathbb{R}, \tag{3.4}$$

where the ij couplings have to follow the connectivity of qubits in the hardware. Measuring the performance of quantum annealing compared to classical annealing schemes is a non-trivial problem, and although advantages of the quantum schemes have been demonstrated in the literature mentioned above, general statements about speed-ups are still controversial [77, 78].

Another famous quantum computational model is *one-way* or *measurement-based quantum computation*. The idea [79] is to prepare a highly entangled state called a *cluster state* and to perform a series of single-qubit measurements which conditionally depend on the output of former measurements. This computation is of course not unitary. The result can be either the state of the unmeasured qubits, or the outcome of a final measurement [80]. Many important quantum algorithms have been implemented using one-way computation [81, 82, 83, 84]. However, I do not know of any quantum machine learning approach based on this model, and it is an open question whether it offers a particularly suitable framework to approach supervised pattern recognition.

3.3 Strategies of information encoding

Recall once more that every quantum system of n qubits can be described by a unit-length 2^n dimensional vector of complex numbers $\mathbf{a} \in \mathbb{C}^{2^n}$ which represent the amplitudes. In Dirac formulation, the amplitudes appear as the coefficients of the computational basis states,

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle. \tag{3.5}$$

Also recall that every quantum evoulution of a closed system can be described by a unitary operator $U|\psi\rangle$ or alternatively by a Hamiltonian defining a unitary operator e^{-iHt} (if H is time-independent).

[3]See http://www.dwavesys.com/ [January 2017]

Classical object	Properties	quantum state
Basis encoding		
$(b_1, ..., b_d),\ b_i \in \{0, 1\}$	b encodes $\mathbf{x} \in \mathbb{R}^N$ in binary	$\|\mathbf{x}\rangle = \|b_1, ..., b_d\rangle$
Amplitude encoding		
$\mathbf{x} \in \mathbb{R}^{2^n}$	$\sum_{i=0}^{2^n-1} \|x_i\|^2 = 1$	$\|\psi_\mathbf{x}\rangle = \sum_{i=0}^{2^n-1} x_i \|i\rangle$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^m}$	$\sum_{i=0}^{2^n-1} \sum_{j=0}^{2^m-1} \|a_{ij}\|^2 = 1$	$\|\psi_\mathbf{A}\rangle = \sum_{i=0}^{2^n-1} \sum_{j=0}^{2^m-1} a_{ij} \|i\rangle\|j\rangle$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	$\sum_{i=0}^{2^n-1} a_{ii} = 1, a_{ij} = a_{ji}^*, \mathbf{A}$ pos.	$\rho_\mathbf{A}$ with $\rho_{ij} = a_{ij}$
Dynamic encoding		
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	U unitary	$U_\mathbf{A}$ with $U_{ij} = a_{ij}$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	$a_{ij} = a_{ji}^*$	$H_\mathbf{A}$ with $H_{ij} = a_{ij}$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^m}$	-	$H_{\tilde{\mathbf{A}}}$ with $\tilde{\mathbf{A}} = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix}$

Table 3.2: A summary of the different types of information encoding presented in the text. This table also summarises the notation of basis, amplitude and dynamic encoded information into the description of a quantum system used in this thesis.

There are different ways to encode information into an n-qubit system. To my knowledge, no terminology to distinguish between such strategies has been proposed yet, and I will therefore refer to them as *basis encoding*, *amplitude encoding*, and *dynamic encoding*. Questions of information encoding become especially important for machine learning and data mining, where information or patterns are extracted from large datasets. The following chapters as well as the conclusion of this thesis will heavily rely on the language developed in this section. Ways of actually encoding datasets in the different representations will be discussed in Chapter 4 and are a crucial part of quantum machine learning algorithms. An illustration of the different encoding methods can be found in Figure 3.3 and a summary of the notation used here can be found in Table 3.2.

3.3.1 Basis encoding

Basis encoding associates a computational basis state of an n-qubit system (such as $|\psi\rangle = |0011\rangle$) with a classical n-bit-string (0011). In a way, this is the most straight forward way of computation, since a bit literally gets replaced by a qubit, and a 'computation' acts on all bit sequences in a superposition in parallel.

The value of the amplitudes of each basis state does not carry any other information than to 'mark' the result of the computation with a high enough probability of being measured. For example, if the basis state $|0011\rangle$ has a probability $|a_{0011}|^2 > 0.5$, repeated execution of the algorithm and measurement of the final state in the computational basis will reveal it as the most likely measurement result, and hence the overall result of the algorithm. For

the basis encoding method, the goal of a quantum algorithm is therefore to increase the probability or absolute square of the amplitude that corresponds to the basis state encoding the solution.

Like in classical computers, basis encoding uses a binary representation of numbers. A quantum state $|x\rangle$ with $x \in \mathbb{R}$ will therefore refer to a binary representation of x with the number n of bits that the qubit register encoding $|x\rangle$ provides. There are different ways to represent a real number in binary form, for example by fixed or floating point representations. In the following it is always assumed that such a strategy is given. The most simple strategy is to define that if x is an integer and

$$x = \sum_{k=0}^{n-1} b_k \frac{1}{2^k}, \tag{3.6}$$

then the binary sequence is $(b_0 \dots b_{n-1})$. If x is in the interval $[0, 1]$, it can be approximated as

$$x = \sum_{k=0}^{n-1} b_k \frac{1}{2^{-k}}, \tag{3.7}$$

and the binary representation is likewise given by the b_k as $(b_0 \dots b_{n-1})$.

Since every classical algorithm can be translated into a reversible routine [85] it can be implemented on a quantum computer.[4] Formally this means that if there is a classical routine that maps $x \to g(x)$ we can find a quantum algorithm or unitary evolution that implements

$$|x\rangle|0\rangle \to |x\rangle|g(x)\rangle.$$

The central problem is to identify quantum algorithms that can bring structural advantages over classical machine learning algorithms, which is of course much more demanding.

3.3.2 Amplitude encoding

Amplitude encoding associates classical numbers with quantum amplitudes, and there are different options to do so. A normalised classical vector $\mathbf{x} \in \mathbb{C}^{2^n}, \sum_k |x_k|^2 = 1$ can be represented by the amplitudes of a quantum state $|\psi\rangle \in \mathcal{H}$ as

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix} \leftrightarrow |\psi_{\mathbf{x}}\rangle = \sum_{j=0}^{2^n-1} x_j |j\rangle.$$

In the same fashion a classical matrix $\mathbf{A} \in \mathbb{C}^{2^n \times 2^m}$ with entries $\sum_{ij} |a_{ij}|^2 = 1$, can be encoded into $|\psi_{\mathbf{A}}\rangle = \sum_{i=0}^{2^m-1} \sum_{j=0}^{2^n-1} a_{ij} |i\rangle|j\rangle$ by enlarging the Hilbert space accordingly. The index registers $|i\rangle, |j\rangle$ refer to the i'th and j'th element respectively. For Hermitian positive trace-1 matrices $\mathbf{A} \in \mathbb{C}^{2^n \times 2^n}$, another option arises: One can associate its entries with the entries of a density matrix $\rho_{\mathbf{A}}$, so that $a_{ij} \leftrightarrow \rho_{ij}$. I will use all three possibilities in Part III.

[4] More precisely, the reversible Toffoli gate is universal for classical Turing machines, and a Toffoli gate can be constructed from elementary quantum gates.

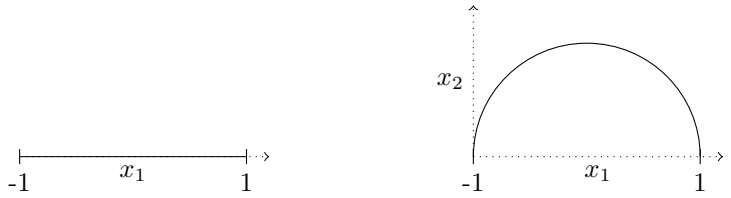


Figure 3.4: Data points in the one-dimensional interval $[-1, 1]$ (left) can be projected onto normalised vectors by adding a constant value in a second dimension x_2 and renormalising.

Encoding information into the probabilistic description of a quantum system necessarily poses severe limitations on which operations can be executed. This becomes particularly important when we want to perform a nonlinear map on the amplitudes, which is impossible to implement in a unitary fashion. This has been extensively debated under the keyword of nonlinear quantum theories [86, 87] and it has been demonstrated that assumptions of nonlinear operators would immediately negate fundamental principles of nature that are believed to be true [88, 89].

Another obvious restriction of this method is that only normalised classical vectors can be processed. Effectively this means that quantum states represent the data in one less dimension or with one less degree of freedom: A classical two dimensional vector (x_0, x_1) is mapped to amplitudes of a qubit (a_0, a_1) which due to $|a_0|^2 + |a_1|^2 = 1$ lie on a unit circle, a one-dimensional shape in two dimensional space. Three dimensional vectors encoded in three amplitudes of a 2-qubit quantum system (where the last of the four amplitudes is redundant and set to zero) will reduce the space to the surface of a sphere, and so on. A remedy can be to increase the space of the classical vector by one dimension $x_{N+1} = 1$ and normalise the resulting vector. The N dimensional space will then be embedded in a $N+1$ dimensional space in which the data is normalised without loss of information.

A special type of amplitude encoding looks at a discrete probability distribution $p = \{p_0, ..., p_{N-1}\}$ that is represented by a quantum state

$$|\psi_p\rangle = \sum_{i=0}^{N-1} \sqrt{p_i}|i\rangle.$$

Such a state is called a *qsample* since measuring the qsample in the computational basis is equivalent to classical sampling from p.

3.3.3 Dynamic encoding

For some applications it can be useful to encode matrices into the dynamic evolution, for example into unitary operators. While unitary operators obviously restrict the class of matrices that they can represent, a useful option from Ref. [90] is to associate a Hamiltonian H with a square matrix $\mathbf{A}$. In case $\mathbf{A}$ is not Hermitian, one can sometimes use the trick of instead encoding

$$\tilde{\mathbf{A}} = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix}. \tag{3.8}$$

and only considering part of the output. This way, the eigenvalues of $\mathbf{A}$ can be processed in a quantum routine, for example to multiply $\mathbf{A}$ or $\mathbf{A}^{-1}$ with an amplitude encoded vector. Another option arises when we look at the dynamics of a subsystem taken out of a larger system with a unitary evolution, $\sum_l K_l \rho K_l^\dagger$. The matrices K_l only have the constraint that $\sum_l K_l^\dagger K_l = 1$. Table 3.2 summarises the different strategies of information encoding mentioned here.

It is interesting to note that many quantum algorithms - such as the matrix inversion routine introduced below - can be understood as strategies of transforming information from one kind of encoding to the other.

3.4 Important quantum routines

The quantum machine learning algorithms in later chapters will make use of some well-known (and a few less well-known) quantum routines that shall be introduced briefly here. Each routine will be summarised in a Box at the end of the subsection. I will also discuss the asymptotic complexity - the growth of the number of elementary operations needed with the input - of the routines, a concept which will be introduced more rigorously in Section 5.1.

3.4.1 Grover search/ Amplitude amplification

Grover's algorithm (here used synonymously with the term 'amplitude amplification') is a quantum routine that finds one or multiple entries in an unstructured (i.e., arbitrarily sorted) database of N entries in basis encoding, a task that on classical computers takes N operations at worst and $N/2$ on average. In more general, it is a routine that given a quantum state in superposition increases the amplitude of some desired basis states, which is a crucial tool for quantum computing. To illustrate this, imagine one had a 3-qubit register in uniform superposition that serves as an index register, joint with a 'flag' ancilla qubit in the ground state as well as the database entries e_i in basis encoding,

$$\begin{aligned}|\psi\rangle =& a_0|000\rangle|0\rangle|\mathrm{e}_0\rangle + \\ & a_1|001\rangle|0\rangle|\mathrm{e}_1\rangle + \\ & a_2|010\rangle|0\rangle|\mathrm{e}_2\rangle + \\ & a_3|011\rangle|0\rangle|\mathrm{e}_3\rangle + \\ & a_4|100\rangle|0\rangle|\mathrm{e}_4\rangle + \\ & a_5|101\rangle|0\rangle|\mathrm{e}_5\rangle + \\ & a_6|110\rangle|0\rangle|\mathrm{e}_6\rangle + \\ & a_7|111\rangle|0\rangle|\mathrm{e}_7\rangle.\end{aligned}$$

Further assume that there was a quantum algorithm known that 'marks' the desired output 001 of the computation by setting the ancilla to 1. This could be a quantum version of a classical routine that analyses an entry and flags it if it is recognised as the correct one, with the typical quantum

property of applying it in parallel to an exponential amount of entries. The result is state

$$\begin{aligned}|\psi'\rangle =& a_0|000\rangle|0\rangle|\mathrm{e}_0\rangle + \\ & a_1|001\rangle|0\rangle|\mathrm{e}_1\rangle + \\ & a_2|010\rangle|0\rangle|\mathrm{e}_2\rangle + \\ & a_3|011\rangle|\mathbf{1}\rangle|\mathrm{e}_3\rangle + \\ & a_4|100\rangle|0\rangle|\mathrm{e}_4\rangle + \\ & a_5|101\rangle|0\rangle|\mathrm{e}_5\rangle + \\ & a_6|110\rangle|0\rangle|\mathrm{e}_6\rangle + \\ & a_7|111\rangle|0\rangle|\mathrm{e}_7\rangle.\end{aligned}$$

Grover search is an iterative quantum algorithm that increases the desired amplitude a_3 so that $|a_3|^2 \approx 1$ and a measurement reveals the result of the computation. It turns out that in order to increase the amplitude, one requires $\sqrt{N}$ iterations (where $N = 2^n$ is the number of basis states or amplitudes over which to 'search') and that this is a lower bound for quantum algorithms for this kind of task [91, 92]. In other words, for search in unstructured databases (a very generic search and optimisation problem) we will not see an exponential speed-up of quantum computers as Grover search is optimal. This is not surprising; if amplitude amplification could be done in exponential time, we could answer the question whether there is a right solution and solve NP-hard problems at a wimp.

One iteration of the Grover routine consists of the following steps:

1. Mark the desired state using the oracle and multiply the amplitude by -1
2. Apply a Hadamard transform on the index qubits
3. Apply a phase shift of -1 on every computational basis state but the first one ($|0...0\rangle$)
4. Apply a Hadamard transform on the index qubits

The third step implements an operator $2|0\rangle\langle 0| - 1$ and together with the Hadamards an 'inversion about the average' is performed. Steps 2-4 make up the so called 'Grover operator' which alters each amplitude according to

$$a_i \rightarrow -a_i + 2\bar{a},$$

where $\bar{a} = \frac{1}{N}\sum_i a_i$ is the average of all amplitudes. After a number of steps proportional to $\sqrt{N}$ the probability of measuring the state of the marked amplitude is maximised if only one state has been marked. For several marked states the final probability depends on their number B and the optimal result (probability of one to measure one of the marked states) will be achieved after a number of steps proportional to $\sqrt{N/B}$.

A helpful analysis and extension of Grover search for quantum superpositions that are not uniform is given in [93] and shows that the distance of unmarked states to the average of all unmarked states remains constant in every iteration, and the same is true for marked states. In other words, the average gets shifted periodically. Also, they find that although one can show that the optimal probability is obtained after $\sqrt{N/B}$ iterations, the value of that probability can

Routine 3.4.1: Amplitude Amplification/Grover Search

Goal: Increase the amplitude of a 'marked' basis state in a uniform quantum superposition.
Input: A quantum system with index register in superposition (entangled with the entries e_i) and an oracle or mechanism to mark the desired amplitude.
Output: The same quantum system in a state where the desired amplitude is 1.
Runtime: $\mathcal{O}(\sqrt{N})$
Variations: Multiple marked amplitudes, non-uniform initial superposition.

****

Require: $\frac{1}{\sqrt{N}}\sum_i^N a_i^t\ |i\rangle|e_i\rangle$

1: **for** $t = 1, ..., \sqrt{N}$ **do**
2: $\quad\frac{1}{\sqrt{N}}\sum_{i=1}^N (-1)^{\delta_{ik}}\ a_i^t\ |i\rangle|e_i\rangle$ ▷ After marking state k
3: $\quad\frac{1}{\sqrt{N}}\sum_{i=1}^N -(-1)^{\delta_{ik}}\ a_i^t + 2\bar{a}^t\ |i\rangle|e_i\rangle$ ▷ After the Grover operator
4: **end for**
5: **return** Measure index register and return the entry e_k

vary considerably depending on the initial distribution of the amplitudes. One therefore needs to pay attention when working with non-uniform initial distributions. If the number of states that are searched for is not known one can use the technique of *quantum counting* to get an estimate [94].

3.4.2 Quantum phase estimation

Quantum phase estimation is a routine that writes information encoded in the phase φ of an amplitude $a = |a|e^{i\varphi}$ into a basis state by making use of the quantum Fourier transform. It is used extensively in quantum machine learning algorithms to extract eigenvalues of operators that contain information about a training set.

3.4.2.1 Discrete Fourier transform

The quantum Fourier transform implements a discrete Fourier transform on the values of the amplitudes. As a reminder, the classical version of the Fourier transform maps a real vector $x \in \mathbb{R}^{2^n}$ to another vector $y \in \mathbb{R}^{2^n}$ via

$$y_k = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{2\pi i j k/2^n} x_j \quad k = 1, ..., 2^n, \tag{3.9}$$

using $\mathcal{O}(n2^n)$ steps. The quantum version maps a quantum state with amplitudes encoding the x_k to a quantum state whose amplitudes encode the y_k,

$$\sum_{j=0}^{2^n-1} x_j|j\rangle \to \sum_{k=0}^{2^n-1} \underbrace{\frac{1}{\sqrt{2^n}}\left[\sum_{j=0}^{2^n-1} e^{2\pi i j k/2^n} x_j\right]}_{y_k} |k\rangle,$$

in only $\mathcal{O}(n^2)$ steps. We need $n = m + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits to get the first m bits of the binary string encoding the eigenstate correct with probability $1-\epsilon$. A quantum Fourier transform applied to a state that is only in one computational basis state $|j\rangle$ (which corresponds to the classical case in which all but one x_j are zero) makes the sum over the j vanish and leaves

$$|j\rangle \rightarrow \sum_{k=0}^{2^n-1} \underbrace{\frac{1}{\sqrt{2^n}} \left[\sum_{j=0}^{2^n-1} e^{2\pi i j k/2^n} x_j \right]}_{y_k} |k\rangle. \tag{3.10}$$

In the quantum phase estimation procedure, the reverse of this transformation will be used.

3.4.2.2 Estimating phases

Given a unitary operator expressed by a Hamiltonian, $U = \mathrm{e}^{\mathrm{i}Ht}$ acting on n_2 qubits, which, applied to one of its eigenstates $|\phi\rangle$ will reveal the corresponding eigenvalue $U|\phi\rangle = \mathrm{e}^{2\pi \mathrm{i}\,\varphi}|\phi\rangle$. The goal is to get an estimate of φ [95].

In order to use the quantum Fourier transform, one needs a unitary transformation $\tilde{U}$ that can implement powers of U conditioned on another index register of $n_1 = n - n_2$ qubits,

$$\sum_{k=1}^{2^{n_1}} |k\rangle|\phi\rangle \rightarrow \sum_{k=1}^{2^{n_1}} |k\rangle U^k|\phi\rangle.$$

Of course, this routine effectively applies U for k times, and unless U reveals some special structure, to stay efficient, k has to be of the order or smaller than n_2. The routine to implement powers of U always has to be given with the algorithm using quantum phase estimation. Use this transformation on an index register in superposition:

$$\begin{aligned}
\frac{1}{\sqrt{n_1}} \sum_{k=0}^{2^{n_1}-1} |k\rangle|\phi\rangle \rightarrow & \frac{1}{\sqrt{n_1}} \sum_{k=0}^{2^{n_1}-1} |k\rangle U^k|\phi\rangle \\
= & \frac{1}{\sqrt{n_1}} \sum_{k=0}^{2^{n_1}-1} |k\rangle(\mathrm{e}^{2\pi \mathrm{i}\, k\varphi}|\phi\rangle) \\
= & \frac{1}{\sqrt{n_1}} \sum_{k=0}^{2^{n_1}-1} \mathrm{e}^{2\pi \mathrm{i}\, k\varphi}|k\rangle|\phi\rangle
\end{aligned}$$

Note that the $|\phi\rangle$ register is not entangled with the rest of the state and can therefore be discarded without any effect. The next step is to apply the inverse quantum Fourier transform on the remaining index register. Consider first the case that $\varphi = \frac{j}{2^{n_1}}$ for an integer j, or in other words, j can exactly be represented by n_1 binary digits. The inverse quantum Fourier transform from Equation (3.10) can be applied and leaves us with state $|j\rangle$. More precisely, $j = b_0 \frac{1}{2^0} + \ldots + b_s \frac{1}{2^s}$ has a s-bit binary representation which is at the same time the binary fraction representation of $\varphi = b_0 \frac{1}{2^0} + \ldots + b_s \frac{1}{2^{-s}}$ (see Equations 3.7, 3.6).

Routine 3.4.2: Quantum phase estimation

Goal: Reveal the eigenvalue φ of an operator U to an eigenstate $|\phi\rangle$ with $U|\phi\rangle = \mathrm{e}^{2\pi \mathrm{i}\,\varphi}|\phi\rangle$.
Input: A quantum state $|0...0\rangle|\phi\rangle$ with n_1 qubits for the first ("index") register and n_2 qubits for the eigenstate register, as well as a quantum routine $\tilde{U}$ that implements powers of U, the operator whose eigenvalue is to be estimated.
Output: A quantum state that approximates $|\varphi\rangle$ with error $\epsilon = \mathcal{O}(1/2^{n_1})$, where φ is encoded into the computational basis state.
Runtime: $\mathcal{O}(n_1^2)$

****

Require: $|0...0\rangle|\phi\rangle$, $\tilde{U}$

1: $\frac{1}{\sqrt{2^{n_1}}} \sum_{k=0}^{2^{n_1}-1} |k\rangle|\phi\rangle$ ▷ After Hadamard on index register

2: $\frac{1}{\sqrt{2^{n_1}}} \sum_{k=0}^{2^{n_1}-1} \mathrm{e}^{2\pi \mathrm{i}\, ik\varphi}|k\rangle|\phi\rangle$ ▷ After $\tilde{U}$

3: $\frac{1}{\sqrt{2^{n_1}}} \sum_{k=0}^{2^{n_1}-1} \mathrm{e}^{2\pi \mathrm{i}\, k(\varphi - \frac{j}{2^{n_2}})}|k\rangle|\phi\rangle \approx |\varphi\rangle|\phi\rangle$ ▷ After inverse quantum Fourier transform

4: **return** State of eigenstate register.

In general, $\varphi \neq \frac{j}{2^{n_2}}$, and the inverse quantum Fourier transform will result in

$$\frac{1}{\sqrt{n}} \sum_{k=0}^{2^{n_2}-1} \mathrm{e}^{2\pi \mathrm{i}\, k(\varphi - \frac{j}{2^{n_2}})}|i\rangle|\phi\rangle.$$

The probability distribution over the computational basis states

$$p_j = \left|\mathrm{e}^{2\pi \mathrm{i}\, k(\varphi - \frac{j}{2^{n_2}})}\right|^2,$$

depends on the difference between φ and the binary fraction representation of integer j. The more accurate or 'close' the binary fraction representation corresponding to integer j is, the higher the probability of measuring the basis state $|j\rangle$ [95]. In a sense, the result is therefore a distribution over different binary representations of φ, and the smaller our error by representing it with n_2 bits is, the narrower the variance of the distribution around the correct representation. This is what is meant when saying that the resulting state of the quantum phase estimation algorithm 'approximates' the phase. The resources needed for quantum phase estimation are the resources to implement powers of U as well as those to implement the quantum Fourier transform.

3.4.3 Postselective amplitude update

Another important quantum routine used frequently in quantum machine learning (for example in the matrix inversion algorithm introduced below, or for quantum state preparation) can adjust the amplitudes by postselection. Unlike Grover search and quantum phase estimation, it is not in the usual canon of quantum algorithms, but it proves useful in the following to introduce and name this routine upfront.

Take a uniform superposition of an index register of $\log(N)$ qubits joined with a register of τ qubits in the ground state and another ancilla qubit

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|0...0\rangle|0\rangle.$$

Assume a quantum routine that writes real numbers in basis encoding into the second register

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|x_i\rangle|0\rangle,$$

where $x_i \leq 1$. For example, such a routine could be a quantum random access memory or a quantum phase estimation algorithm. Now rotate the ancilla qubit conditional on the $|x_i\rangle$ register to get

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|x_i\rangle(\sqrt{1-|x_i|^2}|0\rangle + x_i|1\rangle). \tag{3.11}$$

The details of this step depend on how x_i is encoded into the register. If we assume binary fraction encoding, such a conditional rotation could be implemented by τ single-qubit conditional rotations on the ancilla that are controlled by the $q_1...q_\tau$ qubits of the second register. For qubit q_k the rotation would turn the amplitude closer to $|1\rangle$ by a value of $\frac{1}{2^k}$.

Now the ancilla has to be measured to 'extract the desired branch' of the superposition. If the measurement results in $|1\rangle$ we know that the resulting state is in

$$\frac{1}{\sqrt{N p_{\text{acc}}}} \sum_{i=0}^{N-1} x_i|i\rangle|x_i\rangle|1\rangle,$$

where p_{acc} is the success probability that renormalises the state. Discarding the last register and uncomputing the last but one (which is possible because the superposition is 'kept up' by the index register), we get a state that amplitude encodes a vector $\mathbf{x} = (x_0, ..., x_{N-1})^T$. Of course, one could also start with a non-uniform superposition and would get a product of the initial and new amplitude. If the measurement results in $|0\rangle$ we repeat the entire routine. This is called a *conditional measurement* or *postselection*, which is why I call the entire routine an amplitude update by postselection.

Of course, this procedure is non-unitary and for runtime considerations the likeliness of success has to be taken into account. The probability for the measurement to give the desired result is $p_{\text{acc}} = \frac{1}{N}\sum_i |x_i|^2$ while it fails with $p_{\text{rej}} = \frac{1}{N}\sum_i 1-|x_i|^2$. Relatively uniform distributions over the $\{x_i\}$ close to 1 have a larger probability to succeed. The opposite extreme example is a vector $\mathbf{x}$ in which all entries are zero except from one. In this case, the probability of a successful conditional measurement is exponentially small in the number of qubits. This is of course prohibitive if the algorithm is supposed to run linear in the number of qubits.

Routine 3.4.3: Postselective Amplitude Update

Goal: Transfer values from basis encoding into amplitude encoding.
Input: A uniform superposition of an $n = \lceil \log N \rceil$ index register entangled with N values x_i in basis encoding as well as ancilla register.
Output: A state with amplitudes proportional to x_i.
Runtime: Requires on average $\mathcal{O}(n(\frac{1}{\sqrt{N}}\sum_{i=1}^{N}|x_i|^2)^{-1})$ operations

****

Require: A state $|0...0\rangle|0...0\rangle|0\rangle$ and oracle to load x_i into second register

1: $\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle|x_i\rangle|0\rangle$ ▷ After Hadamard on index register and oracle

2: $\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle|x_i\rangle(\sqrt{1-|x_i|^2}|0\rangle + x_i|1\rangle)$ ▷ After conditional rotation of ancilla

3: $\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}x_i|i\rangle|x_i\rangle|1\rangle$ ▷ After successful conditional measurement

4: **return** $\frac{1}{\sqrt{Np_{\text{acc}}}}\sum_{i=0}^{N-1}x_i|i\rangle$

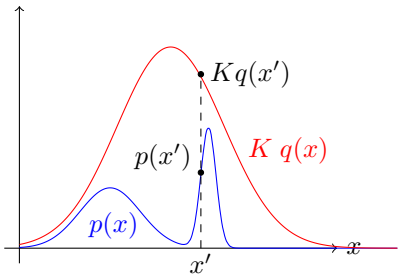


Figure 3.5: Illustration of rejection sampling. In order to sample from the desired distribution $p(x)$ one samples x' instead from a (everywhere bigger) distribution $K\ q(x)$ and accepts the sample with a probability depending on the proportion of the two distributions at that point.

3.4.4 Quantum rejection sampling

There is a close relationship between postselective amplitude update and (classical) rejection sampling. Rejection sampling is a method to draw samples from a distribution $p(x)$ that is difficult to sample from, but for which the probabilities at each point x are easy to evaluate. The idea is to use a distribution $q(x)$ from which it is easy to draw samples (i.e., a Gaussian or uniform distribution), and scale it such that $p(x) \leq Kq(x)$ for an integer $K < \infty$ (see Figure 3.5). One then draws a sample x' from $q(x)$ as well as a random number $u' \sim [0, 1]$. One accepts the sample if $u'Kq(x') < p(x')$ and rejects it otherwise. In other words, the chance to accept the sample from $q(x)$ depends on the ratio of the original and the alternative distribution.

Obviously, if $Kq(x')$ and $p(x')$ are almost equal, the acceptance probability is very high, but in regions where $p(x') << Kq(x')$ we will almost never accept. When repeating this procedure, the accepted samples x' are *effectively* drawn from $p(x)$. The general probability of acceptance can be

calculated as

$$\begin{aligned} p(\text{accept}) &= \mathbb{E}[\text{prob}(u < \frac{p(x)}{Kq(x)})] \\ &= \mathbb{E}[\frac{p(x)}{Kq(x)}] \\ &= \int \frac{p(x)}{Kq(x)} q(x) dx \\ &= \frac{1}{K} \int p(x) dx = \frac{1}{K}, \end{aligned}$$

which shows that K is an important figure of merit for the runtime [96]. The larger it is, or the bigger the difference between the two distributions, the more samples are rejected.

In a postselective amplitude update, we have a quantum state of the form

$$\sum_{i=0}^{N-1} \sqrt{a_i}|i\rangle(\sqrt{1-b_i}|0\rangle + \sqrt{b_i}|1\rangle),$$

where from Equation (3.11) the uniform coefficients $\frac{1}{\sqrt{N}}$ were replaced by more general amplitudes $\sqrt{a_0}, ..., \sqrt{a_{N-1}}$, and the value x_i was replaced by $\sqrt{b_i}$ to avoid confusion with the classical notation. For ease of comparison with classical probability theory, let us assume that the a_i and b_i are all real numbers in $[0, 1]$. By measuring the $|i\rangle$ register we sample from the distribution $\mathcal{A} = \{a_0, ..., a_{N-1}\}$, but our goal is to sample from the 'unknown' distribution $\mathcal{B} = \{a_0 b_0, ..., a_1 b_{N-1}\}$. The conditional measurement on the ancilla therefore plays the role of the rejection step if we associate $q(x) \leftrightarrow \mathcal{A}$ and $p(x) \leftrightarrow \mathcal{B}$. To see this, consider the probability b_i of measuring the ancilla in $|1\rangle$ for state $|i\rangle$ and compare it to the 'rejection sampling' probability of success for sample i using the two distributions, $p(\text{success}_i) = p(ua_i < a_i b_i) = p(u < b_i) = b_i$. The overall probability of acceptance is

$$\begin{aligned} p(\text{accept}) &= \mathbb{E}[p(u < b_i)] \\ &= \mathbb{E}[b_i] \\ &= \sum_i b_i a_i, \end{aligned}$$

where the expectation value $\mathbb{E}$ is taken under the distribution $\mathcal{A}$. This is equivalent to the probability of a successful postselective amplitude update.

Note that this equivalence bears an important argument when comparing quantum and classical algorithms: If it is classically easy to sample from $\mathcal{A}$ and to compute the b_i, a postselective amplitude update can be replaced by classical rejection sampling. The former is not necessarily given, as the example of "quantum supremacy" for boson sampling illustrates [97].

Closely related to the postselective amplitude update is the *quantum rejection sampling* scheme proposed by Ozols et al. [98]. It is based on the idea that measuring a quantum state is equivalent to sampling from the distribution that the quantum state implies. As mentioned before, measuring

the computational basis of the "qsample"

$$\sum_{i=1}^{N} \sqrt{p_i}|i\rangle,$$

with $p_i \in [0,1]$ samples from the discrete probability distribution $p = \{p_1 ... p_N\}$. The difficulty lies in preparing the distribution, for which one may prepare an easy (i.e. factorising) distribution first and then use a postselective amplitude update to arrive at the desired state.

3.4.5 Matrix multiplication and inversion

The previous two routines can be used to multiply a matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ to a vector $\mathbf{x} \in \mathbb{R}^N$ in amplitude encoding, and with a procedure to invert eigenvalues, this can be extended to matrix inversion. This is a rather technical quantum routine which is the heart piece of one approach to quantum-enhanced machine learning. To understand the basic idea, it is illuminating to write $\mathbf{Ax}$ in terms of $\mathbf{A}$'s eigenvalues λ_r and eigenvectors $\boldsymbol{\chi}_r$, $r = 1...R$. The vector $\mathbf{x}$ can be written as a linear combination of eigenvectors of $\mathbf{A}$,

$$\mathbf{x} = \sum_{r=1}^{R} (\boldsymbol{\chi}_r^T \mathbf{x}) \boldsymbol{\chi}_r.$$

Applying $\mathbf{A}$ leads to

$$\mathbf{Ax} = \sum_{r=1}^{R} \lambda_r (\boldsymbol{\chi}_r^T \mathbf{x}) \boldsymbol{\chi}_r. \tag{3.12}$$

and applying $\mathbf{A}^{-1}$ yields

$$\mathbf{A}^{-1}\mathbf{x} = \sum_{r=1}^{R} \lambda_r^{-1} (\boldsymbol{\chi}_r^T \mathbf{x}) \boldsymbol{\chi}_r. \tag{3.13}$$

Under certain conditions, quantum algorithms can find eigenvalues and eigenstates of unitary operators exponentially fast [99]. This promises powerful tools, but one will see that it can only be used for specific problems. I will introduce the matrix multiplication algorithm and then mention how to adapt it slightly in order to do matrix inversion as proposed in [90] (also called the *quantum linear systems of equation* routine).

Consider a quantum state $|\psi_{\mathbf{x}}\rangle$ that represents a normalised classical vector in amplitude encoding. The goal is to map this quantum state to a normalised representation of $\mathbf{Ax}$ with

$$|\psi_{\mathbf{x}}\rangle = \sum_{r=1}^{R} \langle \psi_{\boldsymbol{\chi}_r} | \psi_{\mathbf{x}} \rangle |\psi_{\boldsymbol{\chi}_r}\rangle \rightarrow |\psi_{\mathbf{Ax}}\rangle = \sum_{r=1}^{R} \lambda_r \langle \psi_{\boldsymbol{\chi}_r} | \psi_{\mathbf{x}} \rangle |\psi_{\boldsymbol{\chi}_r}\rangle.$$

This can be done in three steps. First, create a unitary operator $U = \mathrm{e}^{-i\mathbf{A}t}$ and apply it to $|\psi_{\mathbf{x}}\rangle$. Second, do the phase estimation procedure to write the eigenvalues of $\mathbf{A}$ into a register in basis encoding. Third, use a postselective amplitude update to write the eigenvalues into the amplitude.

1. Simulating $\mathbf{A}$

In the first step, we need a unitary operator whose eigenvalue equations are given by $U|\psi_{\boldsymbol{\chi}_r}\rangle = \mathrm{e}^{-i\lambda_r t}|\psi_{\boldsymbol{\chi}_r}\rangle$. If one can evolve a system by e^{iHt} with the Hamiltonian $H = \mathbf{A}$,

this is in fact the case. To resemble a Hamiltonian, $\mathbf{A}$ has to be Hermitian, but the trick from Equation (3.8) circumvents this requirement by doubling the Hilbert with one additional qubit. Techniques to implement general H are called *Hamiltonian simulation* and are discussed in Section 4.3.

The first step prepares a quantum state of the form

$$\frac{1}{K}\sum_{k=1}^{K}|k\rangle e^{-i\mathbf{A}\Delta t}|\psi_{\mathbf{x}}\rangle = \frac{1}{K}\sum_{k=1}^{K}|k\rangle\sum_{r=1}^{R}e^{-i\lambda_r\Delta t}\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|\psi_{\boldsymbol{\chi}_r}\rangle,$$

where on the right side, $|\psi_{\mathbf{x}}\rangle$ is simply expressed in $\mathbf{A}$'s basis as defined above. This is a slight simplification of the original proposal, in which the index register $|k\rangle$ is not in a uniform superposition to exploit some further advantages, but the principle remains the same.

2. Extracting the eigenvalues
In the second step, the quantum phase estimation routine is applied to the index register $|k\rangle$ to 'reduce' its superposition to basis states encoding the eigenvalues,

$$\frac{1}{K}\sum_{r=1}^{R}\sum_{k=1}^{K}\alpha_{k|r}\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|k\rangle|\psi_{\boldsymbol{\chi}_r}\rangle.$$

As explained for the quantum phase estimation routine, the coefficients lead to a large probability $|\alpha_{k|r}|^2$ for computational basis states $|k\rangle$ that approximate the eigenvalues λ_r well. If enough qubits n_1 are given in the $|k\rangle$ register, one can assume that approximately

$$\sum_{r=1}^{R}\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|\lambda_r\rangle|\psi_{\boldsymbol{\chi}_r}\rangle,$$

where $|\lambda_r\rangle$ encodes a n_1 qubit approximation of λ_r. The time needed to implement this step is in $\mathcal{O}(\frac{1}{\epsilon})$ [90].

3. Adjusting the amplitudes
The third step 'writes' the eigenvalues into the amplitudes by using the technique of the postselective amplitude update,

$$\sum_{r=1}^{R}\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|\lambda_r\rangle|\psi_{\boldsymbol{\chi}_r}\rangle(\sqrt{1-\lambda_r^2}|0\rangle+\lambda_r|1\rangle)\rightarrow\sum_{r=1}^{R}\lambda_r\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|\psi_{\boldsymbol{\chi}_r}\rangle,$$

where on the right side and after the successful conditional measurement, the eigenvalue register and ancilla were discarded. Note that this is only possible because no terms in the sum interfere as a result (one could say that the superposition is 'kept intact' by the $|\psi_{\boldsymbol{\chi}_r}\rangle$ state). The final state corresponds to a normalised version of $\mathbf{Ax}$ in amplitude encoding.

For matrix inversion, the last step is slightly adjusted: When conditionally rotating the ancilla qubit, one writes the inverse of the eigenvalue $1/\lambda_r$ into the respective amplitude. Since there is no guarantee that these are smaller than 1, a normalisation constant C has to be introduced that is

of the order of the smallest eigenvalue, and the conditional measurement yields

$$\sum_{r=1}^{R}\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|\tilde{\lambda_r}\rangle|\psi_{\boldsymbol{\chi}_r}\rangle(\sqrt{1-\frac{C^2}{\lambda_r^2}}|0\rangle+\frac{C}{\lambda_r}|1\rangle)\rightarrow C\sum_{r=1}^{R}\frac{1}{\lambda_r}\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|\psi_{\boldsymbol{\chi}_r}\rangle.$$

As discussed above, the conditional measurement is a non-unitary operation and requires the routine to be repeated on average $\mathcal{O}(\frac{1}{p_{\text{success}}})$ times until it succeeds. For matrix multiplication, the success probability is given by $p_{\text{success}}=\sum_r|\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|^2\lambda_r^2$ while for the inversion technique $p_{\text{success}}=\sum_r|\langle\psi_{\boldsymbol{\chi}_r}|\psi_{\mathbf{x}}\rangle|^2\frac{C^2}{\lambda_r^2}\leq\kappa^{-2}$, where κ is the condition number of the matrix defined as the ratio of the largest and the smallest eigenvalue. The condition number is a measure of how 'invertible' or 'singular' a matrix is. Just like when considering numerical stability in classical inversion techniques, when the condition number is large, the quantum algorithm takes a long time to succeed on average.

Example 3.4.1: Simulation of the quantum matrix inversion routine

Since the matrix inversion routine is rather technical, the following classical simulation shall illuminate how a quantum state gets successively manipulated. The matrix $\mathbf{A}$ and vector $\mathbf{b}$ considered here are given by

$$\mathbf{A}=\begin{pmatrix}2/3 & 1/3\\ 1/3 & 2/3\end{pmatrix},\qquad \mathbf{b}=\begin{pmatrix}0.275\\ 0.966\end{pmatrix}.$$

The eigenvalues of $\mathbf{A}$ are $\lambda_1=1$ and $\lambda_2=1/3$, with the corresponding eigenvectors $\mathbf{v}_1=(1/\sqrt{2},1/\sqrt{2})^T$ and $\mathbf{v}_2=(-1/\sqrt{2},1/\sqrt{2})^T$. The number of qubits for the eigenvalue register is chosen as $\tau=10$, and the binary representation of the eigenvalues then becomes $\lambda_1=1111111111$ and $\lambda_2=0101010101$. The error of this representation is <0.001.

The code of the simulation can be downloaded on my *github repository*[5]. The program prints only the basis states of non-zero amplitudes. The qubits are divided in three registers: The first qubit is the ancilla used for the postselective amplitude update, the following ten qubits form the eigenvalue register, and the last qubit initially encodes the vector $\mathbf{b}$, and in the end it encodes the solution $\mathbf{x}$.

State preparation: Encode $\mathbf{b}$ into the last qubit:

```
  Index    Amplitude  Basis State
-------  -----------  --------------
      0     0.275     |0 0000000000 0>
      1     0.966     |0 0000000000 1>
```

After writing the eigenvalues into the second register via phase estimation:

```
  Index    Amplitude  Basis State
-------  -----------  --------------
    682    -0.343     |0 0101010101 0>
    683     0.343     |0 0101010101 1>
   2046     0.618     |0 1111111111 0>
   2047     0.618     |0 1111111111 1>
```

After rotating the ancilla conditioned on the eigenvalue register:

```
  Index    Amplitude  Basis State
```

[5] See https://github.com/mariaschuld/PhD-thesis.

```
-------  -----------  --------------
    682    -0.297     |0 0101010101 0>
    683     0.297     |0 0101010101 1>
   2046     0.609     |0 1111111111 0>
   2047     0.609     |0 1111111111 1>
   2730    -0.172     |1 0101010101 0>
   2731     0.172     |1 0101010101 1>
   4094     0.103     |1 1111111111 0>
   4095     0.103     |1 1111111111 1>
```

After uncomputing the eigenvalue register:

```
  Index    Amplitude  Basis State
-------  -----------  --------------
      0    0.312      |0 0000000000 0>
      1    0.907      |0 0000000000 1>
   2048   -0.069      |1 0000000000 0>
   2049    0.275      |1 0000000000 1>
```

After successful conditional measurement of the ancilla in 1:

```
  Index    Amplitude  Basis State
-------  -----------  --------------
   2048    -0.242     |1 0000000000 0>
   2049     0.970     |1 0000000000 1>
```

The amplitudes now encode the result of the computation, $\mathbf{A}^{-1}\mathbf{b}$:

```
RESULTS: -------------------------------------

Result of the quantum algorithm x =  [-0.412  1.648]

Classical Solution (linalg.solve): x =  [-0.412  1.648]
Classical Solution (manual SVD): x =  [-0.412, 1.648]
```

The comparison to python's 'linalg.solve' package as well as a singular value decomposition confirm this result. The code shows beautifully how the routine starts with a small superposition (here of only two basis states) that gets 'blown up' and then again reduced to encode the two-dimensional output.

3.4.6 Interference circuits

Interference circuits are based on the idea of interfering different branches of a superposition. Just as in wave mechanics, negative values in the amplitudes cancel with positive values and change the resulting amplitude distribution dramatically. Interference is often mentioned as a major resource of quantum computation as opposed to classical computation, and it has been proposed that many quantum algorithms can be formulated as interference algorithms [100]. Here I am mostly interested in the effects of interference to do arithmetic operations on vectors encoded into the amplitudes of quantum states, namely their addition and inner product.

Routine 3.4.4: SWAP test

Goal: Estimate the absolute value of the inner product of two quantum states.
Input: The two quantum states and an ancilla qubit, altogether taking n qubits.
Output: The probability of measuring the ancilla in state 0, $p_{\text{acc}} = 0.5 + 0.5|\langle\psi_x|\psi_y\rangle|^2$, from which the absolute value can be extracted.
Runtime: $\mathcal{O}(n)$

****

Require: Quantum system in state $|0\rangle|\psi_x\rangle|\psi_y\rangle$
1: $\frac{1}{\sqrt{2}}(|0\rangle|\psi_{\mathbf{x}}\rangle|\psi_{\mathbf{y}}\rangle + |1\rangle|\psi_{\mathbf{x}}\rangle|\psi_{\mathbf{y}}\rangle)$ ▷ After Hadamard on ancilla
2: $\frac{1}{\sqrt{2}}(|0\rangle|\psi_{\mathbf{x}}\rangle|\psi_{\mathbf{y}}\rangle + |1\rangle|\psi_{\mathbf{y}}\rangle|\psi_{\mathbf{x}}\rangle)$ ▷ After SWAP operator on states
3: $\frac{1}{2}\left(|0\rangle \otimes (|\psi_x\rangle|\psi_y\rangle + |\psi_y\rangle|\psi_x\rangle) + \frac{1}{2}|1\rangle \otimes (|\psi_x\rangle|\psi_y\rangle - |\psi_y\rangle|\psi_x\rangle)\right)$ ▷ After Hadamard on ancilla
4: **return** Probability of measuring the ancilla in state 0.

3.4.6.1 Amplitude addition by interference

Addition of amplitudes of two quantum states (which of course includes subtraction) is a generic property of quantum systems through interference. However, interference happens between parts of a superposition that describe one and the same quantum system, and one cannot simply create interference 'between' two distinct quantum systems in the respective states $|\psi_{\mathbf{x}}\rangle$ and $|\psi_{\mathbf{y}}\rangle$. We therefore need a clever way to prepare a state

$$\frac{1}{\sqrt{2}}\left(|0\rangle|\psi_{\mathbf{x}}\rangle + |1\rangle|\psi_{\mathbf{y}}\rangle\right), \tag{3.14}$$

in other words, a superposition which consists of two 'branches' flagged by an ancilla qubit. In order to prepare a state like in Equation (3.14) one has to encode a classical vector $(\mathbf{x}^T, \mathbf{y}^T)^T = (x_1, ..., x_N, y_1, ..., y_N)^T$ into the amplitudes of a quantum state. From here, another Hadamard gate onto the ancilla qubit results in

$$|\psi_{\text{if}}\rangle = \frac{1}{2}\left(|0\rangle \otimes (|\psi_{\mathbf{x}}\rangle + |\psi_{\mathbf{y}}\rangle) + \frac{1}{2}|1\rangle \otimes (|\psi_{\mathbf{x}}\rangle - |\psi_{\mathbf{y}}\rangle)\right). \tag{3.15}$$

Measuring the ancilla qubit and only accepting if it is in state $|0\rangle$ leads to the quantum state $|\psi_{\mathbf{x}+\mathbf{y}}\rangle$ (or, if measuring $|1\rangle$, to the state $|\psi_{\mathbf{x}-\mathbf{y}}\rangle$) up to a (re-)normalisation factor. The probability of accepting is $p_{\text{acc}} = \frac{1}{4}\sum_{i=0}^{2^n-1}|x_i \pm y_i|^2$. The interferometry circuit together with a conditional measurement can thus be used for an addition or subtraction of two classical vectors with a following re-normalisation. However, addition is not efficient if the vectors are close to anti-parallel, while subtraction is not efficient when the vectors are close to parallel, which can pose a serious problem for general calculations.

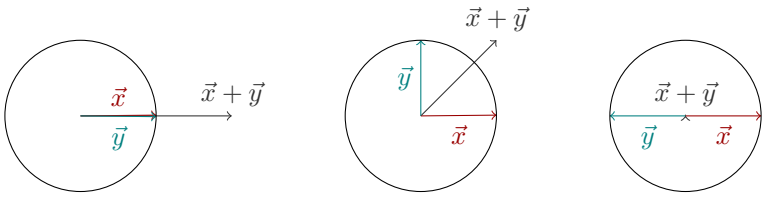


Figure 3.6: Geometric interpretation of the swap test routine. The length of the vector $\mathbf{x}+\mathbf{y}$ is equal to $\sqrt{2+2\mathbf{x}^T\mathbf{y}}$. The sum of the two unit vectors $\mathbf{x}$ and $\mathbf{y}$ therefore contains information on their inner product. This is exploited in the swap test routine.

3.4.6.2 Evaluating inner products of amplitude vectors with the swap test

Although the inner product of two quantum states is an important mathematical object in quantum theory and very useful for algorithms in amplitude encoding, it is not necessarily a trivial operation. The swap routine is a common trick to write the absolute square of the inner product $\langle\psi_\mathbf{x}|\psi_\mathbf{y}\rangle$ into the probability of measuring an ancilla qubit in a certain state. The idea is to create a superposition of an ancilla entangled with the two quantum states we want to evaluate, and swap the quantum states in one of the branches of the superposition to interfere them. Similar to above, this computes a sum of the amplitudes, but for unit vectors there is a close relationship to the inner product: Given two real unit vectors $\mathbf{x}, \mathbf{y}$, then

$$\begin{aligned}\frac{1}{2}(\mathbf{x}+\mathbf{y})^T(\mathbf{x}+\mathbf{y}) &= \frac{1}{2}\sum_i (x_i+y_i)^2 \\ &= \frac{1}{2}\sum_i x_i^2 + \frac{1}{2}\sum_i y_i^2 + \sum_i x_i y_i \\ &= 1+\mathbf{x}^T\mathbf{y}.\end{aligned}$$

A geometric illustration is given in Figure 3.6.

This relationship is exploited by the following routine. Start with a state $|0\rangle|\psi_\mathbf{x}\rangle|\psi_\mathbf{y}\rangle$ and apply a Hadamard on the ancilla, a conditional swap operator on the two registers $|\psi_\mathbf{x}\rangle, |\psi_\mathbf{y}\rangle$ (for the ancilla being in 1), followed by another Hadamard onto the ancilla. The result is a state

$$|\psi_\text{swap}\rangle = \frac{1}{2}|0\rangle\otimes(|\psi_\mathbf{x}\rangle|\psi_\mathbf{y}\rangle + |\psi_\mathbf{y}\rangle|\psi_\mathbf{x}\rangle) + \frac{1}{2}|1\rangle\otimes(|\psi_\mathbf{x}\rangle|\psi_\mathbf{y}\rangle - |\psi_\mathbf{y}\rangle|\psi_\mathbf{x}\rangle)\,,$$

and the probability $p_\text{acc} = 0.5 + 0.5|\langle\psi_\mathbf{x}|\psi_\mathbf{y}\rangle|^2$ of measuring the ancilla in 1 ('acceptance') reveals the absolute value of the inner product through $\sqrt{2p_\text{acc}-1} = |\langle\psi_x|\psi_y\rangle|$. Later other tricks will be presented to retrieve the inner product together with its sign. In the more general case that the two input states are mixed states ρ_x and ρ_y, the result is $p_\text{acc} = \frac{1}{2} + \frac{1}{2}\text{tr}\{\rho_x\rho_y\}$ [101]. Note that here the expression $\rho_x\rho_y$ is not an abbreviation for a tensor product, but the dot product.

The quantum routines described in this chapter are important tools for quantum-assisted machine learning and will be referred to frequently in later chapters.

Part II

Quantum machine learning

While the preceding part introduced foundations from the disciplines of machine learning and quantum information processing separately, Part II will present methods and summarise the literature related to the attempt of combining the two in order to perform supervised pattern recognition on a quantum computer.

Proposals that merge quantum physics and machine learning have been sporadically put forward since quantum computing established itself as an independent discipline. Most notably, a large number of contributions with claims of a 'quantum neural network' model have been published from 1995 onwards (a summary can be found in our review in Ref. [102]). Starting with biological motivations they gradually took a more computer-oriented perspective, but only few of these proposals have rigorous foundations in quantum theory. Questions of learning theory with quantum models (see Section 5.2) came up in the early 2000s. While in 2003, Bonner and Freivals comment in a proceedings of an international workshop of 'Quantum Computation and Learning' that "[q]uantum learning is a theory in the making and its scientific production is rather fragmented" [103] some conclusive results were obtained in the following years [104]. Another early strain of research emerged with the commercialisation of quantum annealing devices by the Canadian company *D-Wave Systems*. Researchers like *Google's* Hartmut Neven thought about how to use these devices for pattern recognition tasks as early as 2008 [105], while more rigorous testing only began in the last couple of years.

The term 'quantum machine learning' came into use around 2013 and predominantly referred to quantum algorithms for machine learning tasks. Lloyd, Mohseni and Rebentrost [106] mention the expression in their (still unpublished) manuscript in 2013, and in 2014, Peter Wittek published an insightful monograph with the title "Quantum Machine Learning. What quantum computing means to data mining" [107], which contains a summary of the earlier contributions. Our own review paper followed in 2015 [108]. Both our review and Wittek's book reflect the still fragmented research landscape at the time.

Only a few years later and with a growing number of contributions, the landscape of quantum machine learning gets more resolved, and the definition is increasingly used in a broader sense of investigations into the synergies of quantum information processing and machine learning in more general. This includes machine learning *for* quantum information processing (see for example [109, 9, 110, 111, 112, 8]) as well as classical computational tools developed within the quantum community that are applied to machine learning (such as the use of tensor networks from many-body physics to train neural networks [5]). An exhaustive list of literature of these approaches has recently been written by Biamonte et al. [113].

The literature on quantum-enhanced or quantum-assisted machine learning, which is of interest in this thesis, starts to form clusters around a couple of distinct approaches. These approaches are essentially based on different tools from quantum information of which some have been presented in the previous section: linear algebra tools based on phase estimation, Grover search, qsampling and quantum annealing. A number of quantum machine learning workshops and conferences[6] have been organised since and witness the formation of a research community.

[6]For example the QML Workshop at the NIPS conference on 12 December 2015, the QML Workshop in South Africa, 18-22 July 2016 and the QML conference at the Perimeter Institute in Waterloo, 8-12 August 2016.

As a summary, the body of literature on quantum machine learning (in both the narrow and broad definition) is growing and diversifying. At this stage we still do not know if the ideas prove strong enough to form a mature research area that can compete with for example the extensive field of quantum cryptography, or if it will be one of many 'fashionable' keywords in science that temporarily gained attention and investments, only to fade away again. The following three chapters will introduce important aspects of the current research landscape from the perspective of my own research. Chapter 4 discusses methods of information encoding into quantum systems, which has been argued to be a shared issue for all quantum machine learning algorithms. Chapter 5 will then solidify the definition of 'quantum enhancement' and summarise some central results from learning theory to set the scene. Finally, Chapter 6 identifies four different categories of quantum machine learning algorithms and provides a review of the existing literature relevant to Part III.

Chapter 4

How to encode datasets into quantum states?

It has been remarked earlier that machine learning algorithms work like funnels that turn large inputs into small outputs. In the extreme case, the output is even a single bit, or a yes-no decision, while the datasets contains millions or even billions of values. Classical machine learning textbooks rarely discuss matters of how to load the data into the processing hardware (although considerations of memory access become important in big data applications). For quantum algorithms, the procedure of writing information into the state (*state preparation*) or dynamics (*quantum simulation*) of a quantum system is not so straight forward, and to ignore it can hide a crucial part of the complexity of an algorithm.[1] This is particularly important as quantum states get destroyed by measurements, and quantum algorithms therefore have to be repeated from the start after every final readout of the result (see also Figure 4.1).

The attitude of different authors towards this issue ranges from simply assuming that the data is given encoded into a quantum system by previous routines, to loosely referring to existing ideas of how to prepare such quantum systems under certain conditions, or, in a few cases, giving more rigorous state preparation routines. What can be stated is that encoding data into quantum systems is a nontrivial problem that requires a lot more attention should quantum machine learning have a chance to be used for realistic applications of supervised learning. The problem is closely related to technological implementations of *quantum memories* and classical-quantum computational interfaces and therefore difficult to address from anything but a merely theoretical perspective at this point, where such technologies are still immature.

In this chapter I want to go through the three encoding methods distinguished in Section 3.3 and discuss strategies of "writing" an arbitrary dataset into a quantum state. These strategies have been partly developed by the quantum machine learning community, and are partly borrowed from quantum information processing research.

Which states we can prepare depends on the technological implementation of the quantum system.

[1] This is not only true for quantum machine learning algorithms. For example the classically hard problem of graph isomorphism is efficiently solvable on a quantum computer if a superposition of isomorph graphs can be created efficiently [114].

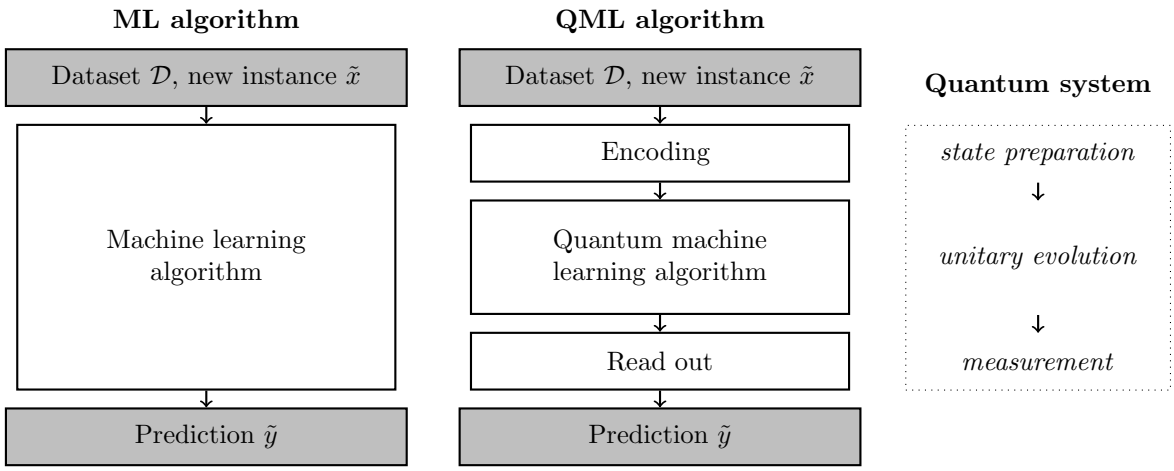


Figure 4.1: In order to solve machine learning tasks based on classical datasets, the quantum algorithm requires an information encoding and read out step that are in general highly nontrivial procedures, and it is important to consider them in the runtime. Published in Reference [115].

.

For example, using a many-body quantum system in equilibrium we naturally get the ground state of the system, while using a photonic setup the same state may be difficult to prepare. A safe assumption when nothing more about the hardware is known is therefore to assume that the n-qubit system starts in the ground state $|0...0\rangle$ and that the data is somehow accessible from a classical memory. Generic state preparation is naturally a nontrivial effort: Any quantum algorithm can be understood as a state preparation routine of the final state before measuring the desired property. State preparation is therefore in a way as difficult as quantum computing itself.

4.1 Basis encoding

Representing a dataset $\mathcal{D} = \{(\mathbf{x}^1, y^1), ..., (\mathbf{x}^M, y^M)\}$ in basis encoding is usually understood as a uniform superposition of basis states that correspond to the inputs (and if necessary entangled with the outputs),

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |\mathbf{x}^m\rangle |y^M\rangle.$$

For example, the two binary input patterns $\{b_1 = (0,1,0,1), b_2 = (1,1,1,0)\}$ with outputs $y_1 = 0, y_2 = 1$ would be encoded as $|\mathcal{D}\rangle = \frac{1}{\sqrt{2}}|0101\rangle|0\rangle + \frac{1}{\sqrt{2}}|1110\rangle|1\rangle$. (In the following I will focus merely on how to encode the inputs, as the output only adds one qubit of complexity). The dataset in basis encoding corresponds to an amplitude vector that has entries $\frac{1}{\sqrt{M}}$ for basis states corresponding to the binary representation of a training vector, and zero entries otherwise. Basis encoded datasets give rise to sparse amplitude vectors, since except in very low dimensions the number of amplitudes $2^{N\tau}$ (where τ is the number of qubits reserved to encode each of the N entries of $\mathbf{x}$) is much larger than the number of nonzero amplitudes M.

An elegant way to construct such 'data superpositions' in time linear in M and N (or, if every feature is encoded in τ qubits, $N\tau$) is given in the works of Ventura and Martinez as well as

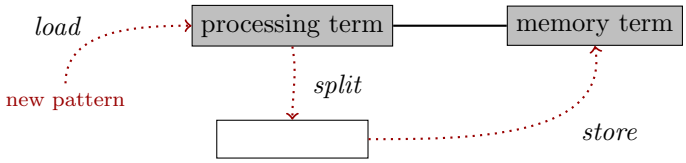


Figure 4.2: Illustration of Ventura and Martinez state preparation routine [116]. The superposition is divided into a processing and memory term, flagged by an ancilla. New training input patterns get successively loaded into the processing term, which gets 'split' by an ancilla, and the pattern gets 'transferred' into the memory term.

Trugenberger [116, 117], and I want to briefly summarise the method here.

Use a quantum system $|l_1, ..., l_N; a_1, a_2; s_1, ..., s_N\rangle$ with three registers, a 'loading' register of N qubits $|l_1, ..., l_N\rangle$, the ancilla register $|a_1, a_2\rangle$ and the storage register $|s_1, ..., s_N\rangle$. As an initial state, prepare $|0_1, ..., 0_N; 0, 1; 0_1, ..., 0_N\rangle$. The algorithm iteratively loads patterns into the loading register and 'breaks away' the right size of terms to add it to the memory branch of the superposition (see Figure 4.2). I will only sketch the m'th iteration here and assume binary features that can be represented by one bit/qubit each. After iterations $1, ..., m-1$ of the algorithm, we obtain the state

$$|\psi^m\rangle = \frac{1}{\sqrt{M}} \sum_{k=1}^{m} |x_1^m, ..., x_N^m; 00; x_1^k, ..., x_N^k\rangle + \sqrt{\frac{M-m}{M}} |x_1^m, ..., x_N^m; 01; 0_1, ..., 0_N\rangle,$$

containing as the first sum the memory term that successively grows as new patterns are written into it and as the second sum the processing term that executes the loading and 'breaking away' process. The m'th pattern $x^m = (x_1^m, ..., x_N^m)$ gets written into the qubits of the loading register. The pattern gets copied into the storage register using an XOR gate, controlled by $a_2 = 1$ (to make sure we only work on the processing term). Using NOT and XOR gates, we compare the qubits of the storage and memory register and flip $a_1 = 1$ if they are the same, which is only true for the processing term. Afterwards apply the unitary

$$cS^p = \begin{pmatrix} \sqrt{\frac{p-1}{p}} & \frac{1}{\sqrt{p}} \\ \frac{-1}{\sqrt{p}} & \sqrt{\frac{p-1}{p}} \end{pmatrix}$$

with $p = M + 1 - m$ controlled by a_1, a_2 being in state 1, which splits the processing term and "adds" the new pattern to the memory term. At last, the comparison steps get undone and the ancilla a_1 reset to zero. Also, the storage register of the processing term gets reset to the ground state, before the next iteration begins. The routine requires $\mathcal{O}(MN)$ steps, and succeeds with certainty.

A slightly faster state preparation routine would be based on Grover search, but requires an oracle that marks the desired basis states. Starting from a uniform superposition of all states, one would amplify the amplitudes of those marked states and would end up in the desired data superposition after $\mathcal{O}(\sqrt{N/M})$ Grover iterations.

Finally, there are interesting proposals for architectures of *quantum random access memories* that

'load' states into a register in parallel by querying an index register in superposition,

$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M}|m\rangle|0...0\rangle \rightarrow \frac{1}{\sqrt{M}}\sum_{m=1}^{M}|m\rangle|\mathbf{x}^m\rangle.$$

While ideas for architectures realising this access in logarithmic time in the number M of items to be loaded have been brought forward [118], a hardware realising such a step is still to be developed [119, 120].

4.2 Amplitude encoding

Some quantum machine learning algorithms encode the (normalised) dataset into the amplitudes of a quantum state

$$|\psi_{\mathcal{D}}\rangle = \sum_{i=1}^{N}\sum_{m=1}^{M} x_i^m|i\rangle|m\rangle = \sum_{i=1}^{N}|\psi_{\mathbf{x}^m}\rangle|m\rangle. \tag{4.1}$$

This quantum state has NM amplitudes and resembles a classical vector containing all training inputs in a large concatenation. The training outputs can be added in an extra qubit $|y^m\rangle$ entangled with the $|m\rangle$ register.

The main advantage is that it allows the construction of algorithms with only logarithmic runtime dependency on the input dimension and/or dataset size. For example, the quantum Fourier transform manipulates the qubits polynomial in n, so that the 2^n amplitudes yield the desired result. This is truly astonishing considering that the basis-2 logarithm of one billion is merely thirty! Such promises sound strange to machine learning practitioners, because simply loading the NM features from the memory hardware takes time that is of course linear in NM. And indeed, the promise of an exponential speedup only holds if state preparation can also be done in time polynomial in the number of qubits n [121], which I will refer to here as *super-efficient*. This is in fact possible in some cases, which the example of preparing a uniform superposition with n Hadamard gates clearly demonstrates. However, there are subspaces in a Hilbert space that cannot be reached from a given initial state [122], which the Grover limit illustrates beautifully. It is therefore an important and nontrivial open question which classes of relevant states for machine learning can be prepared efficiently. Similar caution is necessary for the readout of all amplitudes, which is often again linear in M, N. If the result of the computation is encoded in one amplitude a_i only, the number of measurements needed to retrieve it are of the order of $\mathcal{O}(1/|\mathbf{a}_i|^2)$.

4.2.1 State preparation

One, if not the major, challenge of amplitude encoding is the question of how to prepare an arbitrary quantum state $|\psi\rangle = \frac{1}{\sqrt{2^n}}\sum_{i=0}^{2^n-1} a_i|i\rangle$, $\sum_i |a_i|^2 = 1$, for example to encode a data set. Of particular interest are preparation routines that run polynomial in the number of qubits. I will focus the discussion on real amplitudes. Note that when we separate a complex amplitude $\alpha_i = \mathrm{e}^{i\phi_i}a_i$ into a phase factor $\mathrm{e}^{i\phi_i}$ and a nonnegative real number a_i, if we know how to prepare $\sum_i a_i|i\rangle$ the final state can be constructed by applying small phase rotation gates, approximately

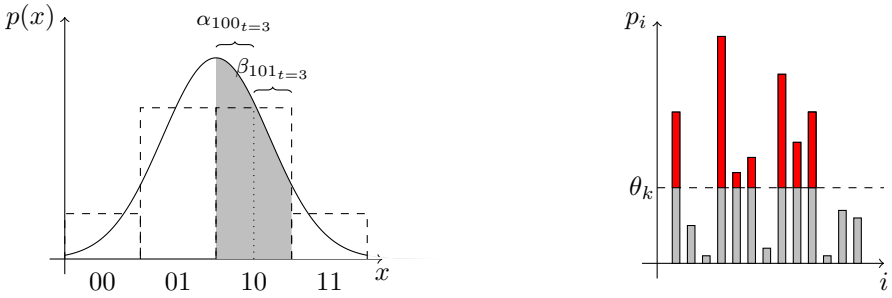


Figure 4.3: Illustration of two super-efficient state preparation routines. Left: Grover and Rudolph's state preparation algorithm for efficiently integrable probability distributions [125]. After step $t = 2$ the domain is distinguished into four regions $i_2 = 00, 01, 10, 11$ of size $\Delta x = 1/2^2$ and amplitudes $p_{00}^{(2)}, p_{01}^{(2)}, p_{10}^{(2)}, p_{11}^{(2)}$. In the third step, the i_2nd region (here demonstrated for $i_2 = 2$) gets split into two. The parameters α_{2_3} and β_{2_3} are the probability to find a random variable in the left or right part of the region. This procedure successively prepares a finer and finer discretisation of the probability distribution. Right: In the k'th step of the routine of Soklakov and Schack [124], an oracle is applied to mark the states whose probabilities p_i are larger than a certain threshold θ_k (here in red). These states are amplified with the Grover iterator, resulting in a state that looks qualitatively like the red bars only. In the next step, the threshold is lowered to include the states with a slightly smaller probability in amplitude amplification, until the desired distribution of the p_i is prepared.

taking $\sum_i a_i|i\rangle \to \sum_i a_i \mathrm{e}^{i\phi_i}|i\rangle$ (see also [123, 124]).

In the gate model of quantum computing, a number of proposals have been brought forward to prepare specific classes of states 'super-efficiently'. For example, Grover and Rudolph suggest a scheme that is linear in the number of qubits for the case that we know an efficiently integrable one-dimensional probability distribution $p(x)$ of which the state is a discrete (i.e. coarse-grained) representation [125]. Iteratively, all n qubits are rotated such that after the last step the desired qsample $\sum_i \sqrt{p_i}|i\rangle$ is prepared, where the index i marks the interval $[i\Delta x, (i+1)\Delta x]$ with $\Delta x = \frac{1}{2^n}$. In step t the t'th qubit $|0_t\rangle$ is rotated,

$$\sum_{i_t=0}^{2^t-1} \sqrt{p_i^{(t)}}|i_t\rangle|0_t\rangle \to \sum_{i_{t+1}=0}^{2^{t+1}-1} \sqrt{\alpha_{i_t}}|i_t\rangle|0_t\rangle + \sqrt{\beta_{i_t}}|i_t\rangle|1_t\rangle,$$

such that α_{i_t}, $[\beta_{i_t}]$ are the probabilities for a random variable to lie in the left [right] half of region of the i_t'th interval of a coarse graining into 2^t regions of size $\Delta x = \frac{1}{2^t}$ (see Figure 4.3 left). With each step, this process prepares an increasingly fine discretisation of the probability distribution $p(x)$. Essentially the same idea was proposed at a similar time by Kaye and Mosca [123], who do not refer to probability distributions but demand in general that the conditional probability $p(q_k = 1|q_1...q_{k-1}) = \left(\frac{\alpha_{q_1,\ldots,q_{k-1},1}}{\alpha_{q_1,\ldots,q_{k-1}}}\right)^2$, which is the chance that given the state $q_1...q_{k-1}$ of the previous qubits, the k'th qubit is in state 1 (or the random variable falls in the right region), is easy to compute. A state for which calculating the conditional probabilities is easy is a superposition of basis states of a certain number of nonzero qubits. However, for most states $p(q_k = 1|q_1...q_{k-1})$ entails a number of probabilities that is exponential in k.

Soklakov and Schack [124] propose an efficient scheme to approximately prepare quantum states whose amplitudes $a_i = \sqrt{p_i}$ represent a discrete probability distribution p_i, $i = 1, ..., 2^n = N$, and

all probabilities p_i are of the order of $1/\eta N$ for $0 < \eta < 1$. The algorithm then runs polynomial in the number of qubits and in the inverse parameters $\eta^{-1}, \epsilon^{-1}, \nu^{-1}$ and produces a result that with probability greater than $1 - \nu$ has an error smaller than ϵ. To sketch the basic idea (Figure 4.3 right), a series of oracles are defined that mark states i for which p_i are larger than a certain threshold, and with each oracle this threshold is lowered by a constant amount (controlling the maximum error of the final state). The states marked by k'th oracle include the states marked by any oracle before, and increases their amplitude with Grover iterations. The probability distribution is successively shaped this way. Note that to know the optimal number of Grover iterations, quantum counting has to be applied to estimate the number of marked states in the current run.

Under the condition that the states to prepare are sufficiently uniform, a more generic approach is to refer to a quantum random access memory, or technological implementations that allow to access information 'in superposition', by querying an index register [90, 126, 127, 128]. More precisely, a quantum random access memory implements the query $1/\sqrt{N}\sum_i |i\rangle|0\rangle \to 1/\sqrt{N}\sum_i |i\rangle|x_i\rangle$, retrieving the basis encoded quantum states $|x_i\rangle$. A postselective amplitude update can then prepare the state $\sum_i x_i|i\rangle$ (see also [129]). However, the price to pay is that the entire routine has to be repeated if the conditional measurement fails, and the probability of success $p_{\text{success}} = \sum_i |x_i|^2/N$ is obviously dependent on the state to encode. While a uniform distribution of the x_i ensures that $p_{\text{success}} = \sum_i \frac{1}{2N} = \frac{1}{2}$, a sparse vector puts a lot of weight into the 'failure branch' of the ancilla to be measured, and in the extreme of only one nonzero entry we get $p_{\text{success}} = \frac{1}{N}$ which is exponentially small in n. In other words, we would have to repeat the measurement $\mathcal{O}(N)$ times on average to prepare the correct state. Luckily, very sparse states can be prepared by other methods. (For example, for a one-sparse vector one can simply flip the qubit register from the ground state into a basis state representing the index i.) Zhao et al. [130] therefore propose that in case of s sparse vectors, one does not apply the quantum random access memory to a uniform superposition, but a superposition only of the basis states representing nonzero indices,

$$\frac{1}{\sqrt{s}} \sum_{i|x_i \neq 0} |i\rangle.$$

There are many other possible ways to prepare quantum states beyond quantum circuits. An interesting perspective is offered by Aharanov et al. [114]. They present the framework of "adiabatic state generation" as a natural (and polynomially equivalent) language to analyse state preparation in the gate model. The idea is to perform an adiabatic evolution of a quantum system initially in the ground state of a generic Hamiltonian to the ground state of a final Hamiltonian, which is the final state we wish to prepare. This translates the question of which initial states can be easily prepared to the question of which ground states of Hamiltonians are in reach of adiabatic schemes, i.e. have small spectral gaps between the ground and the first excited state. They show that given a sequence of simulable (i.e. sparse) Hamiltonians with certain properties (i.e., nonnegligible spectral gaps to ensure the transition time is polynomial, strong enough overlap between the ground states of two consecutive Hamiltonians), then there is an efficient quantum algorithm to reach the desired ground state. They furthermore provide techniques to efficiently generate states connected to certain classes of Markov chains.

A similar idea is to use the unique stationary states of a dissipative process in an open quantum system allow for a generic method of state preparation. Of course, as in adiabatic state preparation the time to reach this state is crucial. For Markovian evolutions described by a Gorini-Kossakowski-Sudarshan-Lindblad master equation it can be shown that for any $|\psi_{\text{in}}\rangle$ there is a master equation with a relaxiation time $T_{\text{relax}} \propto 1/(\gamma_l)_{\text{min}}$ where $(\gamma_l)_{\text{min}}$ is the smallest of the dissipation rates or couplings in the master equation [131]. Another challenge is to engineer an environment that obeys the desired dynamic equations.

In summary, quantum state preparation for amplitude encoded states relevant to machine learning algorithms is a topic that still requires a lot of attention, and claims of exponential speedups should therefore be viewed with a pinch of skepticism.

4.2.2 Readout

In amplitude encoding, the amplitudes themselves contain the result of a computation. One way to access all amplitudes of a n-qubit quantum state is to execute a full *quantum state tomography* to retrieve the $2^n \times 2^n$ entries of the corresponding density matrix. Again, only a single amplitude cannot be 'read out' super-efficiently in general, since it indicates the probability of a measurement result which can be exponentially small in the number of qubits. Many proposals for approximate tomography have been made, some of which even consider methods of machine learning [112, 132]. But to remain efficient regarding the number of qubits, one has to apply clever tricks to encode the desired outcome in statistical information that can be extracted by only a few measurements. In quantum machine learning, this is often done by a swap test and its variations, which reveals the inner product of two quantum states.

4.3 Dynamic encoding

Encoding matrices that contain the data into operators that define the evolution of a quantum system allows us to extract eigenvalues of such matrices, or to multiply them with a vector in amplitude encoding. Of particular interest for the matrix inversion algorithm in Section 3.4.5 as well as related quantum machine learning algorithms based on linear algebra calculus (Section 6.1, Chapters 7, 8) are strategies where the dataset is encoded into the Hamiltonian $H_{\mathbf{A}}$. The goal is then to apply an evolution $\mathrm{e}^{iH_{\mathbf{A}}t}$ to a given quantum state $|\psi\rangle$ of n qubits. Extensive research in the area of *quantum simulation* has been done to tackle the problem of how to 'simulate' an arbitrary Hamiltonian H (an excellent review can be found in [133]), and without going into the details the major results relevant here shall be summarised in the following.

4.3.1 Hamiltonian simulation

An important result from the theory of so called *digital Hamiltonian simulation* is that there is an efficient routine for a universal quantum computer to simulate s-sparse Hamiltonians of n-qubit systems (the runtime is linear in s, n) [134, 135, 136]. This ensures a logarithmic ('super-efficient') dependency on the dimension of the matrix $\mathbf{A}$ we wish to encode into the Hamiltonian. It was also shown that this is essentially optimal. The dependence on the inverse error has recently been improved to $\mathcal{O}(\log \frac{1}{\log \epsilon})$. For nonsparse Hamiltonians, efficient simulation is in general not

Routine 4.3.1: Quantum simulation of sparse Hamiltonians

Goal: Implement $\mathrm{e}^{-iHt}|\psi\rangle$ for arbitrary s-sparse H with maximum error ϵ and a given n qubit system $|\psi\rangle$.
Input: Quantum state $|\psi\rangle$ and an oracle that accesses entries of the desired Hamiltonian H.
Output: $|\psi'\rangle = \mathrm{e}^{-iHt}|\psi\rangle$
Runtime: $\mathcal{O}(s\ n\ \log \frac{1}{\log \epsilon})$

possible [137].

To sketch the theory behind this result (see for example [138, 133]), one has to break up the evolution of H for time t into small steps of Δt,

$$\mathrm{e}^{iHt} = (\mathrm{e}^{iH\Delta t})^{\frac{t}{\Delta t}}. \tag{4.2}$$

If the Hamiltonian can be expressed as a sum of Hamiltonians with local interactions $H = \sum_{k=1}^{K} H_k$, then $\mathrm{e}^{iH\Delta t}$ can be decomposed into local gates that are much easier to implement,

$$\mathrm{e}^{iH\Delta t} = \prod_k \mathrm{e}^{iH_k\Delta t} + \mathcal{O}(\Delta t^2).$$

Of course, there is a trade off, as the smaller Δt, the smaller the error of this decomposition, but the more often the sequence has to be repeated in Equation (4.2). Note that the decomposition may be nontrivial [139].

An important alternative method used in later chapters works for low-rank Hermitian matrices $\mathbf{A}$ and has been recently proposed as a technique for a quantum version of principal component analysis [140]. The basic idea is to use a simulation of a swap operator S to effectively exponentiate a nonsparse density matrix. This is based on the formal relation

$$\mathrm{tr}_2\{e^{-iS\Delta t}(\sigma \otimes \rho)e^{iS\Delta t}\} = e^{-iH_\rho\Delta t} + \mathcal{O}(\Delta t^2). \tag{4.3}$$

In words, simulating the swap operator that 'exchanges' the state of the qubits of state ρ and σ for a short time Δt, and taking the trace over the second quantum system results in an effective dynamic as if exponentiating the second density matrix. To prove the relation, write the operator-valued exponential functions as series and apply the swap operators, using that $S^2 = 1$ (see Appendix A.2). The error of the approximation is in $\mathcal{O}(\Delta t^2)$ which is negligible for sufficiently small simulation times Δt. The effective Hamiltonian is entry-wise equivalent to ρ which is possible because H and ρ are both Hermitian. A swap operator is sparse and its simulation therefore efficient. For example, the swap gate for two qubits reads

$$S_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Routine 4.3.2: Density matrix exponentiation (simple)

Goal: Simulate $\mathrm{e}^{-iH_\rho t}\ |\psi\rangle\langle\psi|\ \mathrm{e}^{iH_\rho t}$ for a time t where H_ρ is entrywise equal to the density matrix $\rho = |\psi\rangle\langle\psi|$.
Input: Two quantum systems in the state $\rho = |\psi\rangle\langle\psi|$.
Output: An approximation with error ϵ of the pure state $|\psi'\rangle = \mathrm{e}^{-iH_\rho t}|\psi\rangle$
Runtime: $\mathcal{O}(\frac{t^2}{\epsilon})$

An important adjustment has to be made in order to prepare the state for the inverse quantum Fourier transform in the quantum phase estimation procedure. As presented in Section 3.4.2, one requires a superposition of powers $(e^{-iH_\rho \Delta t})^k$. Lloyd et al. [140] show that this can be done by using 2^d copies of ρ joined with an index register of d qubits in superposition,

$$\sum_{k=1}^{2^d} |k\Delta t\rangle\langle k\Delta t| \otimes \sigma \otimes \rho^{(1)} \otimes \ldots \otimes \rho^{(2^d)}$$

and simulate a sequence of 2-qubit swap operators, each of which swaps the first state σ with the gth copy of ρ. The swap operator sequences are entangled with an index register, so that for index $|k\Delta t\rangle$ the sequence of swap operators runs up to copy $\rho^{(k)}$,

$$\frac{1}{K}\sum_{k=1}^{K} |k\Delta t\rangle\langle k\Delta t| \otimes \prod_{g=1}^{k} e^{-iS_g \Delta t}$$

In Appendix A.2 it is shown in detail that after taking the trace over all copies of ρ, this effectively implements the evolution

$$\sum_{k=1}^{K} |k\rangle\langle k| \otimes e^{-ikH_\rho \Delta t}\sigma\ e^{ikH_\rho \Delta t} + \mathcal{O}(\Delta t^2),$$

which is precisely in the form required to apply the quantum Fourier transform.

Note that the desired accuracy is not necessarily a constant but depends on the size of H_ρ, especially when we are interested in the eigenvalues of ρ. To see this, consider the eigenvalues are all uniform. Since $\mathrm{tr}\rho = 1$, they are all given by $\frac{1}{N}$ if N is the number of diagonal elements or the dimension of the Hilbert space of $|\psi\rangle$. We certainly want the error to be smaller than the eigenvalues themselves, which means that $\epsilon < \frac{1}{N}$, hence, the runtime dependent on N and the number of copies needed for the density matrix exponentiation in superposition depends on N^3. As a consequence, this method is only polynomial in the number of qubits (i.e., 'super-efficient') if the eigenvalues of H_ρ are dominated by a few large eigenvalues that do not require a small error to be resolved. In other words, the matrix has to be well approximable by a low-rank matrix. For matrices containing the dataset, this means that the data is highly redundant.

To summarise this chapter, while not all datasets can be (super-)efficiently encoded into quantum

Routine 4.3.3: Density matrix exponentiation (in superposition)

Goal: Simulate $\sum_k |k\rangle\langle k| \otimes \mathrm{e}^{-ikH_\rho t}|\psi\rangle\langle\psi|\mathrm{e}^{iH_\rho t}$ for a time t where H_ρ is entry-wise equal to the density matrix $\rho = |\psi\rangle\langle\psi|$.
Input: A number of quantum systems in ρ which is of the order of $\mathcal{O}(\frac{1}{\epsilon^3})$.
Output: An approximation with error ϵ of the pure state $|\psi'\rangle = \sum_k |k\rangle \otimes \mathrm{e}^{-ikH_\rho t}|\psi\rangle$
Runtime: $\mathcal{O}(\frac{t^2}{\epsilon})$

states, there are important exceptions for some structures. More research is needed to link these classes to real-life machine learning implementations and to develop further methods extending the current literature.

Chapter 5

How can quantum information improve machine learning?

Provided that we have a way to encode the dataset into properties of a quantum system we can start to design quantum machine learning algorithms. Such algorithms may be interesting as a mere theoretical investigation of how machine learning with quantum information could work. However, the expression quantum-*enhanced* machine learning suggests that the quantum algorithm has to offer some advantages compared to classical methods. (Naturally, the head start of classical computing requires a striking motivation before one considers to invest into quantum technologies that might only be ready for large-scale implementations many decades in the future.) There is more than one measure of merit when it comes to machine learning. I will discuss three dimensions here[1], namely *time complexity*, *sample complexity* and *model complexity*. This serves as a wider literature review as well as a methodological investigation. The last chapter of Part II will then focus on quantum machine learning algorithms that offer runtime improvements, which is the narrow context of what follows in Part III.

5.1 Time complexity

The concept of time complexity has already been used extensively in the previous sections, where it was assumed that the reader is familiar with the basic concepts. It shall be explained in more detail here. The runtime of an algorithm on a computer is the time it takes to execute the algorithm, in other words, the number of elementary operations multiplied by their respective execution times. In conventional computers, the number of elementary operations could in theory still be established by choosing the fastest implementations for every subroutine and count the logic gates. However, with a fast technological advancements in the IT industry it becomes obvious that a device-dependent runtime is not a suitable theoretical tool to measure the general speed of an algorithm. This is why computational complexity theory looks at the *asymptotic complexity* or the rate of growth of the runtime with the size of the input digits. (Asymptotic refers to the fact that one is interested in laws for sufficiently large inputs only.) If the resources needed to execute an algorithm or the number of elementary operations grow polynomially with the size of the input, it is *tractable* and the problem in theory *efficiently solvable*. Exponential

[1] Thanks to a talk by Vedran Dunjko at the Quantum Machine Learning Workshop in South Africa in July 2016.

growth makes an algorithm *intractable* and the problem *hard*.

An illustrative example for an intractable problem is guessing the number combination for the security code of a safe (which, without further structure requires a brute force search): While a code of two digits only requires 100 attempts in the worst case (and half of those guesses in the average case), a code of 10 digits requires ten billion guesses in the worst case, and a code of 30 digits has more possible configurations than our estimation for the number of stars in the universe. No advancement in computer technology could possibly crack this code.

When thinking about quantum computers, estimating the actual runtime of an algorithm is even more problematic. Not only do we not have a unique implementation of qubits yet that gives us a set of elementary operations to work with (as well as their time scales), but even if we decided on a set of elementary gates, it is nontrivial to decompose quantum algorithms into this set.[2] In many cases, we can claim that we know there is such a sequence of elementary gates, but it is by no means clear how to construct it.

Due to the lack of fault-tolerant universal quantum computing, the vast majority of authors in quantum information processing are therefore interested in the asymptotic complexity of their routines, and how they compare to classical algorithms. The field of *quantum complexity theory* has been developed as an extension to classical complexity theory [141, 142] and is based on the question whether quantum computers are in principle able to solve computational problems faster in relation to the time complexity. (In the following, complexity will always refer to the asymptotic runtime complexity unless stated otherwise.) A runtime advantage in this context is called a *quantum enhancement*, or simply a *quantum speed-up*. The term *quantum supremacy* also gained popularity among researchers to refer to applications that no classical computer could possibly compute.

A collaboration of researchers at the forefront of benchmarking quantum computers came up with a useful typology for the term of 'quantum speed-up' [143].

1. A *provable quantum speed-up* requires a proof that there can be no classical algorithm that performs as well or better than the quantum algorithm. A provable speed-up has been demonstrated for Grover's algorithm, which scales quadratically better than classical [64] given the oracle to mark the desired state [144].

2. A *strong quantum speed-up* compares the quantum algorithm with the best known classical algorithm. The most famous example of a strong speed-up is Shor's quantum algorithm to factorise numbers in time growing polynomially (instead of exponentially) with the number of digits of the prime number, which due to far-reaching consequences for cryptography systems gave rise to the first major investments into quantum computing.

3. If we relax the 'best' classical algorithm (which is often not known) to the 'best available' classical algorithm we get the definition of a *common quantum speed-up*.

4. The fourth category of *potential quantum speed-up* relaxes the conditions further by comparing two specific algorithms and relating the speed-up to this instance only.

[2] Thanks to Matthias Troyer for his talk at the Quantum Research Group in October 2016 for this insight.

5. Lastly, and useful when doing benchmarking with quantum annealers, is the *limited quantum speed-up* that compares to "corresponding" algorithms such as quantum and classical annealing.

Although the holy grail of quantum computing (and consequently quantum machine learning) remains to find a *provable exponential speed-up*, the wider definition of a quantum advantage in terms of the asymptotic complexity opens a lot more avenues for realistic investigations. Two common pitfalls have to be avoided. Firstly, quantum algorithms often have to be compared with classical sampling, which is likewise nondeterministic and has close relations to quantum computing (see Section 3.4.4). Secondly, some claim that complexity is hidden in spatial resources, for example the resources to implement a quantum random access memory, and that quantum computers have to be compared to cluster computing, which will surely be its main competitor in years to come [145, 143].

I want to introduce some particulars of how to formulate the asymptotic complexity in relation to quantum machine learning algorithms. The 'size' of the input in the context of machine learning usually refers to the number of data points M as well as the number of features N. When dealing with sparse inputs that can be represented more compactly, the latter is sometimes replaced by the maximum number s of nonzero elements in a training input. The complexity commonly considers other numbers of merit for the analysis: The error ϵ of a value z is the precision to which the corresponding value z' calculated by the algorithm is correct, $\epsilon = ||z - z'||$ (with a suitable norm). When dealing with matrices, the condition number κ, which is the ratio of the largest and the smallest eigenvalue or singular value, is sometimes of interest. Many quantum machine learning algorithms have a chance of failure, for example because of a conditional measurement. In this case the average number of attempts required to execute it successfully needs to be taken into account as well. A success probability of p_s generally leads to a factor of $1/p_s$ in the runtime (for example, if it will only succeed in 1% of the cases, one has to repeat it on average for 100 times).

To express the asymptotic complexity, the runtime's dependency on these variables is expressed in the "big-O" notation (see Figure 5.1):

- $\mathcal{O}(g(n))$ means that the running time has an upper bound of $\lambda g(n)$ for some $\lambda \in \mathbb{R}$ and the input n.
- $\Omega(g(n))$ means that the running time has a lower bound of $\lambda g(n)$ for some $\lambda \in \mathbb{R}$ and the input n.
- $\Theta(g(n))$ means that the running time has a lower bound of $\lambda_1 g(n)$ and an upper bound of $\lambda_2 g(n)$ for some $\lambda_1, \lambda_2 \in \mathbb{R}$ and the input n.

Having introduced time complexity, the question remains what speed-ups can be detected specifically in quantum-enhanced machine learning. Of course, machine learning is based on common computational routines such as search or matrix inversion, and quantum machine learning derives its advantages from tools developed by the quantum information processing community. It is therefore no surprise that the speed-ups achieved in quantum-enhanced machine learning are directly derived from the tool box of quantum information processing. Roughly speaking (and with more details in the next chapter), three different types of speed-ups are claimed:

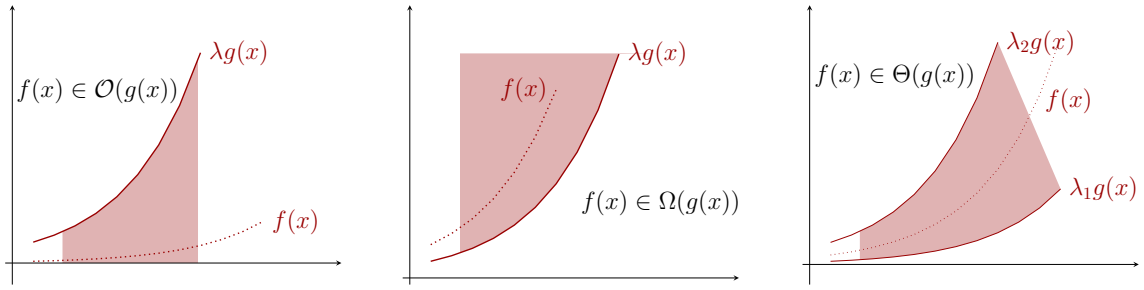


Figure 5.1: Illustration of the big-O notation. If a function $f(x)$ (in this context f is the runtime and x the input) is 'in' $\mathcal{O}(g(x))$, there exists a $\lambda \in \mathbb{R}$ such that $|f(x)| \leq \lambda g(x)$ for large enough x. The Ω symbol stands for the inequality $|f(x)| \geq \lambda g(x)$, while the Θ symbol signifies that there are two $\lambda_1 < \lambda_2 \in \mathbb{R}$ such that $f(x)$ lies between the functions $\lambda_1 g(x)$ and $\lambda_2 g(x)$.

1. Provable quadratic quantum speed-ups derive from variations of Grover's algorithm applied to search problems, as outlined in more detail in Section 6.2 of the following chapter. Learning can always be understood as a search in a space of hypotheses [146]. Examples of such speed-ups are Wiebe et al. [147] who search for discrimination hyperplanes for a perceptron in their representation as points on a sphere, and Low et al. [148] who prepare quantum states that capture the probability distribution of Bayesian nets.

2. Exponential speed-ups are naturally more tricky, even if we only look at the categories of either *strong* or *common speed-ups*. Within the discipline, they are usually only claimed by quantum machine learning algorithms that execute linear algebra computations in amplitude encoding (see Section 6.1). But these come with serious conditions, for example about the sparsity or redundancy of the dataset, which may allow for very fast classical algorithms as well. The question of state preparation furthermore excludes many states from super-efficient preparation.

3. More specific comparisons occur when quantum annealing is used to solve optimisation problems derived from machine learning tasks [149], or to prepare a quantum system in a certain distribution [150]. In summary, it is not clear yet whether any provable speed-ups will occur, but benchmarking against specific classical algorithms show that there are problems where quantum annealing can be of advantage (see Section 6.3 and 6.4). While offering only limited theoretical prediction, these approaches have the advantage to be testable in the lab and serve as vehicles to further our understanding of present-day implementations.

The next chapter will investigate this summary in more detail, which will also be the focus of the original contributions in Part III.

5.2 Sample complexity

Besides the asymptotic complexity, the so called *sample complexity* is an important figure of merit in machine learning. It refers to the number of samples needed to learn a model. Samples can either be given as training instances drawn from a certain distribution (*examples*) or by computing outputs to specific inputs (*queries*). While for the supervised pattern recognition problem defined above the dataset is given as examples, one can easily imagine a slightly different setting where queries are possible, for example when a certain experiment results in input-output pairs, but due

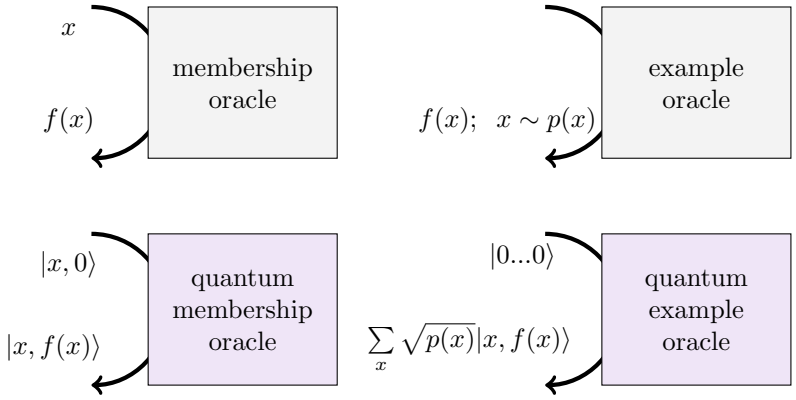


Figure 5.2: Different types of oracles to determine the sample complexity of a learning algorithm. A membership oracle takes queries for a certain input x and returns the value $f(x)$, while an example oracle is activated and draws samples of x from a certain (usually unknown) distribution $p(x)$, again returning the value $f(x)$. The quantum version of a membership oracle is a function evaluation on a register $|x\rangle \otimes |0\rangle \to |x\rangle \otimes |f(x)\rangle$, while a quantum example oracle has been defined as the qsample of the distribution $p(x)$.

to the effort of performing a single experiment one cannot allow for arbitrarily many data points to be generated.

Considerations about sample complexity are usually based on binary functions $f : \{0,1\}^N \to \{0,1\}$. The sample complexity of a machine learning algorithm refers to the number of samples that are required to learn a *concept* from a given *concept class*. A concept is the rule f that divides the input space into subsets of the two class labels 0 and 1, in other words, it is the law that we want to recover with a model.

There are two important settings in which sample complexity is analysed.

1. In *exact learning from membership queries* [151], one learns the function f by querying a membership oracle with inputs x and receives the answer whether $f(x)$ evaluates to 1 or not.

2. The framework of *Probably Approximately Correct (PAC)* learning was introduced by Valiant [152] and asks how many examples from the original concept are needed in the worst case to train a model so that the probability of an error ϵ (i.e., the probability of assigning the wrong label) is smaller than $1 - \delta$ for $0 \leq \delta \leq 1$. The examples are drawn from an arbitrary distribution via an example oracle. This framework is closely linked to the VC-dimension d of a model (see Chapter 2).

To translate these two settings into a quantum framework (see Figure 5.2), a quantum membership oracle as well as a quantum example oracle are introduced. They are in a sense parallelised versions of the classical sample generators, and with quantum interference of amplitudes this parallelism can be used to extract more information. Rather surprisingly, it turns out that the classical and quantum sample complexity are polynomially equivalent, or as stated by Servedio and Gortler [104]:

> [F]or any learning problem, if there is a quantum learning algorithm which uses polynomially many [samples] then there must also exist a classical learning algorithm which uses polynomially many [samples].

Note that this only concerns the sample complexity; the same authors find an instance for a problem that is efficiently learnable by a quantum algorithm in terms of *time complexity*, while the best classical algorithm is intractable. I want to briefly explain this finding in more detail for the two different learning frameworks and their translation to quantum computing.

5.2.1 Exact learning from membership queries

Sample complexity in relation to queries is closely related to the concept of 'quantum query complexity' which is an important figure of merit in quantum computing in the oracular setting (for example to determine the runtime of Grover's algorithm). A quantum oracle can be described as a unitary operation

$$U : |x\rangle \otimes |s\rangle \rightarrow |x\rangle \otimes |s \oplus f(x)\rangle,$$

where $x \in \{0,1\}^n$ is encoded in the computational basis. [3].

Two famous quantum algorithms that demonstrate how for specific types of problems only a single quantum query can be sufficient are the Deutsch-Josza and the Bernstein-Vazirani algorithm. They are both based on the principle of applying the quantum oracle to a register in uniform superposition, thereby querying all possible inputs in parallel. Writing the outcome into the phase and interfering the amplitudes then reveals information on the concept, for example if it was a balanced (half of all inputs map to 1) or constant (all inputs map to 1) function. Note that this does not mean that the function itself is learnt (i.e., which inputs of the balanced function map to 0 or 1 respectively) and is therefore not sufficient as an example to prove theorems on general quantum learnability.

Servedio and Gortler [104] analysed the query complexity in the quantum membership oracle setting and found that if any class C of boolean functions $f\{0,1\}^n \rightarrow \{0,1\}$ is learnable from Q quantum membership queries, it is then learnable by $\mathcal{O}(nQ^3)$ classical membership queries. This result shows that classical and quantum learnability are the same in this framework, with at most a polynomial overhead. Shortly after, Hunziker et al. [154] introduce a larger framework which they call "impatient learning" and proposed the following two conjecture on the actual number of samples required in the (asymptotic) quantum setting:

Conjecture 1: For any family of concept classes $C = \{C_i\}$ with $|C| \rightarrow \infty$, there exists a quantum learning algorithm with membership oracle query complexity $\mathcal{O}(\sqrt{|C|})$.

Conjecture 2: For any family of concept classes $C = \{C_i\}$ with $|C| \rightarrow \infty$, there exists a quantum learning algorithm with membership oracle query complexity $\mathcal{O}(\frac{\log|C|}{\sqrt{\gamma}})$ [where $\gamma \leq 1/3$ is a measure of how easy it is to distinguish between concepts, and small γ indicate a harder class to learn].

The classical upper bound for exact learning from membership queries is given by $\mathcal{O}(\frac{\log|C|}{\gamma})$. While the first conjecture was proven by Ambainis et al. [155], the second conjecture was proven by Atici and Servedio [156] shortly after up to a $\log\log|C|$ factor.

[3] Note that counting 'parallel' function evaluations by an oracle and a classically computable function is not an uncontested comparison [153]

With these results, it becomes apparent that no exponential speed-up can be expected from quantum sample complexity in the way defined above, while quadratic speed-ups are known to be achieved. This generalises the optimality of Grover search and is a fundamental limit to quantum computation.

5.2.2 PAC learning from examples

It is a well-established fact from classical learning theory that a (ϵ, δ)-PAC learning algorithm for a nontrivial concept class C of VC-dimension d (see Section 2.1.2) requires at least $\Omega(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{d}{\epsilon})$ examples, but can be learnt with at most $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{d}{\epsilon}\log\frac{1}{\epsilon})$ examples [157, 158].

A first contribution to quantum PAC learning was made by Bhshouty and Jackson in 1998 [159], who define the important notion of a quantum example oracle. They show that a certain class of functions (so called "polynomial-size disjuncitve normal form expressions") which are actively investigated in the PAC setting are efficiently learnable by a quantum computer from 'quantum example oracles' which draw examples from a uniform distribution. (Note that the PAC model requires learnability under any distribution, so this result is much more specific). The quantum example oracle is equivalent to what has been introduced earlier as a *qsample*,

$$\sum_i \sqrt{p(x)}|x, f(x)\rangle, \tag{5.1}$$

where the probabilities of measuring the basis states in the computational basis reflects the distribution $p(x)$ from which the examples are drawn.[4]

Servedio and Gortler [104] use this definition to show the equivalence of classical and quantum PAC learning in the given framework by proving that if any class C of boolean functions $f\{0,1\}^n \to \{0,1\}$ is learnable from Q evaluations of the quantum example oracle, it is then learnable by $\mathcal{O}(nQ)$ classical examples. Improvements in a later paper by Atici and Servedio [156] prove a lower bound on quantum learning that is close to the classical setting, i.e. that any (ϵ, δ)-PAC learning algorithm for a concept class of VC-dimension d must make at least $\Omega(\frac{1}{\epsilon}\log\frac{1}{\delta} + d + \frac{\sqrt{d}}{\epsilon})$ calls to the quantum example oracle from Equation (5.1). This was again improved by Arunachalam et al. [160] to finally show that in the PAC setting, quantum and classical learners require the same amount of examples.

5.2.3 Introducing noise

Even though the evidence suggests that classical and quantum sample complexity are similar up to at most polynomial factors, an interesting observation derives from the introduction of noise into the model. Noise refers to corrupted query results or examples for which the value $f(x)$ is flipped with probability μ. For the quantum example oracle, noise is introduced by replacing it

[4] Their algorithm interfers the amplitudes of the qsample via a quantum Fourier transform, thereby effectively changing the distribution from which to sample and leaving some question whether the comparison to an external example generator is fair. However, they show that while the quantum example oracle can be simulated by a membership query oracle, this is not true vice versa, i.e. the 'quantum example oracle' cannot be used to perform arbitrary membership queries efficiently (which is clear from the Grover limit, since one would have to transform an arbitrary superposition into a certain measurement of the x query). It seems therefore that the quantum example oracle ranges somewhere between a query and an example oracle.

with a mixture of the original qsample and a corrupted qsample weighted by μ.

In the PAC setting with a uniform distribution investigated by Bshouty and Jackson, the quantum algorithm can still learn the function by consuming more examples, while noise is believed to render the classical problem unlearnable [159]. A similar observation is made by Cross, Smith and Smolin [161], who consider the problem of learning n bit parity functions by membership queries and examples, a task that is easy both classically and quantumly, and in the sample as well as time complexity sense. (Parity functions evaluate to 1 if and only if the number of 1s in the input string is odd). However, they find that in the presence of noise, the classical case becomes intractable while the quantum samples required only grow logarithmically. To ensure a fair comparison, the classical oracle is modelled as a dephased quantum channel of the membership oracle and the quantum example oracle respectively, and as a toy model they consider a slight adaptation of the Bernstein-Vazirani algorithm.

These observations are evidence that a fourth category of potential quantum advantages, the robustness against noise, could be a fruitful avenue for further research.

5.3 Model complexity

The term 'model complexity' is a wide concept that usually refers to the flexibility (also: capacity, richness, expressive power) of a model. For example, a model giving rise to linear decision boundaries is less flexible than a model with more complex decision boundaries. A linear fit in regression is less flexible than a higher order polynomial. Roughly speaking, more complex models have a better chance to fit the training data better.

As laid out in Chapter 2, this does not mean that more flexible models have a higher generalisation power. In fact, they are much more prone to overfitting, which Vapnik's upper bound for the generalisation error quantifies. The bound is given by the sum of the empirical error on the training set ϵ_{emp} and an expression that depends on the VC-dimension d. With probability of at least $1-\delta$ for some $\delta > 0$ the bound is given by [28],

$$\epsilon_{\text{gen}} \leq \epsilon_{\text{emp}} + \sqrt{\frac{1}{M}\left(d\left(\log\left(\frac{2M}{d}\right)+1\right)+\log\left(\frac{4}{\delta}\right)\right)}. \tag{5.2}$$

The VC-dimension d of a model is in fact closely related to the model complexity and is defined for binary models, $f : \mathcal{X} \to \{0,1\}$. It refers to the maximum number of data points M assigned to two classes for which a hypothesis or 'trained version' of the model $f(\mathbf{x};\theta)$ makes no classification error or *shatters* the data. For the next higher number of points $M+1$ there is no dataset that can always be shattered, no matter how the labels are assigned. For example, a linear boundary can shatter three data points, but there are label assignments for which it cannot separate four datapoints (see Figure 5.3).

It is intuitively clear that one prefers a model function that can express the natural dynamics of the system producing the data. For example, nonlinearities in the updating function of recurrent neural networks allow us to represent nonlinear dynamics of the real-world systems they try to

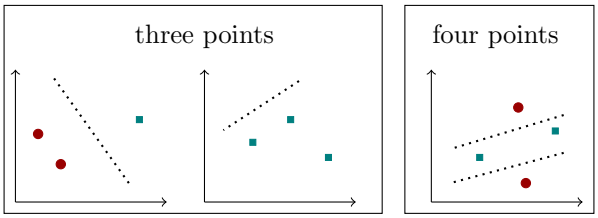


Figure 5.3: A model imposing a linear decision boundary in a two-dimensional input space can separate or shatter a dataset of three points in any label configuration, but there are datasets of four points which it cannot separate.

recover. It is therefore interesting how 'quantum models' might enhance the space of possible dynamics that can be mimicked. This becomes particularly important when the systems producing the data are quantum systems, which has been termed the 'QQ' case in the introduction of this thesis. However, also the 'CQ' case, i.e. classically produced data modeled by a quantum model, may reveal interesting facets, and only few attempts have been made in this direction so far.

One model that we understand fairly well from a classical and quantum perspective is the Ising model, whose dynamics are used in Hopfield networks for associative memory recall. A number of studies on "quantum Hopfield models" investigate how the introduction of an inverse field term changes the dynamics of the model significantly. For example, Inoue [162] introduces quantum fluctuations and analyses image restoration via quantum Monte Carlo simulations, finding that quantum fluctuations give finer restoration results than thermal fluctuations. He later performs a similar study on Hopfield networks under 'quantum noise' [163] joined by others [164], revealing a few more particulars. Many years previously, a study by Nonomura and Nishimori [165] with a similar aim came to the conclusion that with regards to the ratio of stored patterns and system size, "quantum fluctuations play quite similar roles to thermal fluctuations in neural networks". Another advantage can be a drastically increased storage capacity in a quantum annealing framework [166]. Even though the usefulness of these findings to speed up machine learning applications is not necessarily the focus of this kind of work, it raises the interesting point of how the quantum version of a classical model can have different properties.

Another model that lends itself easily to quantum generalisations are hidden Markov models, where a system undergoes a sequence of state transitions under observations, and either the transition/observation probabilities, or the next transition is to be learnt (see Section 2.3.3.2). State transitions and observation probabilities are central to quantum physics as well, where they can be elegantly described in the formalism of density matrices and open quantum systems. Monras, Beige and Wiesner [167] therefore introduce a quantum formulation of hidden Markov models, in which the state of the system is a density matrix ρ and state transition are formally represented by Kraus operators $\mathcal{K}$ with associated probabilities $\mathrm{tr}\{\mathcal{K}\rho\}$. Although the primary idea is to use these quantum models to learn about partially observable quantum systems, they give evidence that the number of internal states necessary for generating some stochastic processes are less than of a comparable classical model.[5] The 'reinforcement learning' version of hidden Markov models are partially observable Markov decision processes and have been generalised

[5] In a later paper, Monras and Winter [168] also perform a deep mathematical investigation into the task of learning a model from example process data in the quantum setting, i.e. to ask whether there is there a quantum process that realises the observations.

to a quantum setting by Barry, Barry and Aaronson [169]. The authors show that a certain type of problem, the existence of a sequence of actions that can reach a certain goal state, is undecidable for quantum models. Another variation of a hidden Markov model is provided by [170] and based on Gudder's quantum Markov processes. Again, the message is that quantising a model can enlarge the framework and make it useful for the analysis of quantum dynamics.

A third branch of 'quantum-extendable' models are graphical models. Leifer et al. [58] for example give a quantum mechanical verison of belief propagation. A number of studies also look at quantum extensions of the closely related causal models. (Causal models can roughly be imagined as directed graphical models, in which a directed edge stands for a causal influence. An increasingly important goal is to discover causal structure from data, which is possible only to some extend [51] in the classical case.) Quantum systems exhibit very different causal properties, and it is sometimes possible to infer the full causal structure of a model from (quantum) data [171, 172]. Costa and Shrapnel [173] introduce quantum analogues of causal models for the discovery of causal structure and propose a translation for central concepts such as the Markov condition or faithfulness. If such quantum models can have any use for classical data is an interesting question.

Other dimensions of quantum models have been investigated. For example, Stark [174] asks whether computers are able to learn optimal (that is, low-dimensional) quantum models from experimental data, and find that like for classical models the problem is NP-hard. Dunjko et al. [4] investigate a general quantum speed-up for quantum enhanced reinforcement learning by modeling agents as quantum systems.

The contributions mentioned do not answer (and in most cases, they do not even directly address) the question of the complexity or flexibility of quantum models, which is a potential topic for further research. Interesting questions may be what VC dimension a quantum channel has if we interpret it as the model function $f(\mathbf{x}, w)$, or if there are dynamics for which a quantum model requires a much lower number of parameters than a classical model. If such quantum models are found, one can think of quantum simulations or special-purpose quantum computers to use as a classification device and investigate potential (classical and quantum) training methods.

In summary, this chapter presented three ways of thinking about quantum-*enhanced* machine learning, namely the time, sample and model complexity. While under the definitions outlined above, sample complexity does not seem to change in the quantum case, interesting aspects concern the robustness against noise as well as the generalisation properties of quantum models. In the following I will focus on the first dimension and analyse the advantages in terms of time complexity that a quantum algorithm offers. However, as a conclusion to this chapter it should be clear that there is a lot more to quantum machine learning than just speed-ups.

Chapter 6

Recent progress in quantum machine learning algorithms

After discussing results for the sample complexity of quantum learning as well as potential avenues to investigate quantum models for machine learning, this chapter will focus on the narrower research question, namely to develop quantum machine learning algorithms that solve supervised pattern recognition tasks with improvements in the *time complexity*. With the lack of a large-scale universal quantum computer, most quantum algorithms remain theoretical proposals, although some proof-of-principal experiments have been executed [175, 176, 177] and some proposals have been tested on the D-Wave machine as a special purpose quantum computer.

I divide the literature into four main approaches (see Table 6.1): Those that compute linear algebra routines 'super-efficiently' with quantum systems (Section 6.1), approaches based on Grover search or quantum walks (Section 6.2), approaches where samples are drawn from quantum states (Section 6.3), and approaches where the solution to an optimisation problem is encoded in the ground state of a Hamiltonian (Section 6.4). These four classes -to my own surprise - cover most of the proposals for quantum machine learning algorithms found in the literature up to today.

6.1 Simulating linear algebra calculus with qubits

The quantum machine learning algorithms presented in this section are rather technical and rely mainly on the subroutines introduced in previous chapters. I will therefore briefly remind of their purpose. Approaches to prepare a quantum state that encodes a classical vector in its amplitudes were outlined in Section 4.2, while Section 4.3 showed how to create an evolution where the Hamiltonian corresponds to a classical matrix which is either of the form of a low-rank approximable density matrix (*density matrix exponentiation*) or a sparse matrix (*Hamiltonian simulation*). Harrow, Hassidim and Lloyd's *quantum matrix inversion* routine from Section 3.4.5 then uses *quantum phase estimation* (Routine 3.4.2) to extract the eigenvalues of this matrix, after which a *postselective amplitude update* (Routine 3.4.3) is used to write the inverted eigenvalues into the amplitudes, effectively executing the matrix inversion. The swap test (Routine 3.4.4) reveals inner products of quantum state vectors. Quantum machine learning algorithms based on linear algebra calculus with quantum systems use different combinations of these tools. They all

Problems	QIP tools	Applied to
Simulating linear algebra calculus with qubits		
- matrix inversion - inner products - eigenvalue decomposition - singular value decomposition	- quantum phase estimation - postselective amplitude update - Hamiltonian simulation - density matrix exponentiation	- support vector machines [126] - Gaussian processes [130] - linear regression [128, 178] - discriminant analysis [179] - recommendation systems [180] - principal component analysis [140]
Optimisation with Grover search		
- finding closest neighbours - Markov chains	- amplitude amplification - quantum walks	- k-nearest neigbour [181] - page ranking [182, 2, 183] - clustering [184] - associative memory [116, 185] - perceptrons [147] - active learning agents [2] - natural language processing [186]
Sampling from quantum states		
- sampling from model distribution	- quantum annealing - quantum rejection sampling	- Boltzmann machines [63, 187, 188, 189, 181] - Bayesian nets [148] - Bayesian inference [190]
Optimisation with ground states of Hamiltonians		
- combinatorial optimisation	- adiabatic quantum computing - quantum annealing - quantum simulation	- associative memory [191, 192, 166] - boosting [105, 149, 193] - debugging [194] - variational Bayes inference [195] - Bayesian networks [196] - perceptron [197] - EM algorithm [198] - clustering [199, 200]

Table 6.1: Summary of four approaches to quantum-enhanced machine learning. Listed are the computational problems they tackle, the tools from quantum information processing (QIP) they use and what type of machine learning method they have been applied to already.

have a runtime that is logarithmic in the input dimension N as well as the training set size M (assuming state preparation is easy and that the rank or sparsity of the matrix is logarithmic in N, M), but have slightly different polynomial dependencies on the desired maximum error and/or the condition number of the matrix. The following presentation will therefore focus on three main points:

1. What is the linear algebra problem the machine learning method requires to solve?

2. How are the subroutines combined and extended to perform pattern recognition?

3. Which preconditions ensure the optimal runtime?

My own contribution to this approach to quantum machine learning can be found in Chapters 7 and 8.

6.1.1 Basic idea

For a number of machine learning algorithms, the mathematical problem of training the model can be reduced to linear algebra routines such as inverting a matrix or finding a matrix' eigenvalues

and eigenvectors. The matrices are usually constructed from the training set and therefore grow with the dimension N and/or number of the training vectors M. The basic idea of the linear algebra approach in quantum machine learning is to use quantum systems for linear algebra calculus, whereby the quantum states represent classical vectors via amplitude encoding, and the Hamiltonian of the system represents a classical matrix via dynamic encoding.

The most straight forward example is a multiplication of a vector and a unitary matrix. The evolution of a quantum system can be mathematically expressed as $\mathbf{Ua}$, where $\mathbf{U}$ is a unitary matrix and $\mathbf{a}$ is the complex vector of 2^n amplitudes. Performing this evolution therefore effectively implements the unitary matrix multiplication, and the state of the quantum system after the evolution encodes the result of the multiplication. But one can do much more: Quantum systems are natural eigendecomposers in the sense that the results of measurements are eigenvalues of operators to certain eigenstates. This can be used for matrix inversion, as demonstrated in the algorithm for linear systems by Harrow, Hassidim and Lloyd [90]. One can say that the dynamics of the quantum system *simulates* a linear algebra calculation with the properties of qubits.

The crux of this idea lies in the fact that many quantum subroutines require a number of manipulations that is polynomial in the number n of qubits. Algorithms composed of such subroutines depend only logarithmic on the number $N = 2^n$ of amplitudes, and the same usually applies to the number of training vectors M. A logarithmic dependency is a significant speedup. For example, a training set of one billion vectors that have each dimension one million can be represented in a quantum state with 10^{15} amplitudes, which can be represented by only 50 qubits. Of course, there has to be a caveat, and this is state preparation. The 10^{15} amplitudes have to be encoded into the quantum system first. Some routines to do this polynomially in the number of qubits have been presented in Chapter 4, but they only work under certain conditions. Still, the promises of simulating linear algebra calculus with quantum systems for big data applications are severe, and worth investigating further.

6.1.2 Inverting data matrices

The first suggestion to use Harrow, Hassidim and Lloyd's (HHL's) quantum matrix inversion technique for statistical data analysis was proposed by Wiebe, Braun and Lloyd [128]. Closely related to machine learning, their goal was to 'quantise' linear regression for data fitting, in which the best fit parameters of a linear function for some data has to be found. In Section 2.3.1.1 it has been shown that basic linear regression (i.e., without regularisation) reduces to finding a solution to the equation

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

where $\mathbf{w}$ is a N-dimensional vector containing the fit parameters, $\mathbf{y}$ is a M-dimensional vector containing the target outputs $y^1, ..., y^M$ from the dataset, and the rows of the 'data matrix' $\mathbf{X}$ are the N-dimensional training inputs $\mathbf{x}^1, ..., \mathbf{x}^M$. The most demanding computational problem behind linear regression is the inversion of the square of the data matrix, $(\mathbf{X}^T\mathbf{X})^{-1}$. Wiebe, Braun and Lloyd assume that $\mathbf{X}$ is sparse (which means that the data consists overwhelmingly of zero entries), and that a quantum state $|\psi_\mathbf{y}\rangle$ representing a normalised version of $\mathbf{y}$ can be prepared efficiently. This allows them to apply the HHL algorithm twice. The first time $\mathbf{X}^T\mathbf{y}$ is

computed following the HHL routine with $\mathbf{A} \to \mathbf{X}^T$ and $\mathbf{b} \to \mathbf{y}$, but instead of extracting the inverted eigenvalues, the original eigenvalues are computed in the postselective amplitude update (as explained in the matrix multiplication routine described in Section 3.4.5). In a second step the original HHL routine for matrix inversion is used with $\mathbf{A}^{-1} \to (\mathbf{X}^T\mathbf{X})^{-1}$ applied to the result of the first step, $\mathbf{b} \to \mathbf{X}^T\mathbf{y}$. The output of the quantum algorithm are the normalised fit parameters encoded in the amplitudes of the quantum state $|\psi_{\mathbf{w}}\rangle$. The authors conclude that "[a]lthough the algorithm yields [the state encoding the parameters] efficiently, it may be exponentially expensive to learn via tomography", and shift their focus to a routine which estimates the quality of a fit.

The runtime of the routine depends on the condition number κ of the data matrix $\mathcal{O}(\kappa^6)$, where the Hamiltonian simulation as well as the conditional measurement in the postselective amplitude update are each responsible for a term $\mathcal{O}(\kappa^3)$. Hamiltonian simulation furthermore contributes a cubic dependency on the sparsity s of the data matrix, but with more recent methods [136], this can be reduced to a linear dependency. The runtime grows with the inverse error as $\mathcal{O}(1/\epsilon)$.

To solve a pattern classification task in machine learning, the quantum algorithm for data fitting has to be extended by a classification step that predicts a new input using the learnt parameters. Given that the vast majority of datasets is nonsparse, it is also desirable to relax the sparsity condition for $\mathbf{X}$, as well as to improve the dependency on the condition number. I will show in Chapter 7 how to formulate a quantum algorithm for linear regression that fulfils these properties for low-rank approximable (i.e., highly redundant) data matrices $\mathbf{X}$ using the density matrix exponentiation technique.

6.1.3 Inverting kernel matrices

Another fertile application is demonstrated in Rebentrost, Mohseni and Lloyd's quantum machine learning algorithm for support vector machines [126]. While support vector machines as presented in Section 2.3.4.3 do not directly lead to a matrix inversion problem, a very popular version called *least-squares support vector machines* [201] turns the convex quadratic optimisation into least squares optimisation. In short, by replacing the inequality constraints in Equation (2.31) with equalities, the support vector machine becomes equivalent to a regression problem of the form

$$\begin{pmatrix} 0 & 1...1 \\ 1 & \\ \vdots & \mathbf{K} \\ 1 & \end{pmatrix} \begin{pmatrix} w_0 \\ \boldsymbol{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{y} \end{pmatrix},$$

with the kernel Gram matrix of entries $(\mathbf{K})_{ij} = (\mathbf{x}^m)^T\mathbf{x}^{m'}$ and the Lagrangian parameters $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_M)^T$. To obtain the α_i one has to invert the kernel matrix. Note that for simplicity I did not include slack parameters as in the original proposal.

Rebentrost et al. develop an elegant trick by recognising that the kernel matrix is proportional to

the partial density matrix of a quantum data state,

$$|\psi_{\mathcal{D}}\rangle = {}^{1}/_{\sqrt{\chi}} \sum_{i=1}^{N} |m\rangle|\psi_{\mathbf{x}^m}\rangle,$$

with the training inputs in amplitude encoding $|\psi_{\mathbf{x}^m}\rangle = \sum_i x_i|i\rangle$. Taking the partial trace of the corresponding density matrix $|\psi_{\mathcal{D}}\rangle\langle\psi_{\mathcal{D}}|$ over the register $|i\rangle$ results in a density matrix that is a normalised version of $\mathbf{K}$,

$$\mathrm{tr_i}|\mathcal{D}\rangle\langle\mathcal{D}| = \frac{1}{\chi} \sum_{m,m'=1}^{N} \langle\psi_{\mathbf{x}^m}|\psi_{\mathbf{x}^{m'}}\rangle|m\rangle\langle m'| = \frac{\mathbf{K}}{\mathrm{tr}\,\mathbf{K}}\,.$$

After preparing the kernel matrix as a density matrix, the authors can use the density matrix exponentiation technique to simulate $\mathrm{e}^{i\mathbf{K}t}$ and to apply $\mathbf{K}^{-1}$ to a quantum state $|\psi_{\mathbf{y}}\rangle$ via the HLL routine. The outcome of the quantum algorithm is a quantum state encoding the parameters $w_0, \alpha_1, ..., \alpha_M$ as

$$|\psi_{w_0,\boldsymbol{\alpha}}\rangle = \frac{1}{w_0^2 + \sum_m \alpha_m^2} \left(w_0|0\rangle + \sum_{m=1}^{M} \alpha_m|m\rangle \right).$$

The remaining classification step is a slight amendment to the swap test. As shown in Equation (2.34), one can compute the desired label $\tilde{y}$ via

$$\tilde{y} = \sum_{m=1}^{M} \alpha_m y^m \mathbf{x}^m \tilde{\mathbf{x}} + w_0.$$

We essentially want to compute the inner product of the new input with an expression combining the training inputs with the found Lagrangian parameters. To achieve this, one first needs to add another qubit register and prepare (via the same state preparation routine used before)

$$|\psi_{\mathbf{u}}\rangle = \frac{1}{w_0^2 + \sum_m \alpha_m^2 |x^m|^2} \left(w_0|0\rangle|0\rangle + \sum_{m=1}^{M} \alpha_m|m\rangle|\psi_{\mathbf{x}^m}\rangle \right),$$

as well as a separate quantum system representing the new input one wants to classify,

$$|\psi_{\tilde{\mathbf{x}}}\rangle = \frac{1}{M|\tilde{\mathbf{x}}|^2 + 1} \left(|0\rangle|0\rangle + \sum_{m=1}^{M} |m\rangle|\psi_{\tilde{\mathbf{x}}}\rangle \right).$$

The swap test will not be performed using $|\psi_{\mathbf{u}}\rangle$ and $|\psi_{\tilde{\mathbf{x}}}\rangle$ as inputs, but with an interference circuit (see Section 3.4.6). Implement all the above routines conditioned on an extra ancilla to get

$$\frac{1}{\sqrt{2}} \left(|0\rangle|\psi_{\mathbf{u}}\rangle + |\psi_{\tilde{\mathbf{x}}}\rangle|1\rangle \right).$$

When performing a measurement in the basis $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, there is a probability of $\frac{1}{2}(1 - \langle\psi_{\mathbf{u}}|\psi_{\tilde{\mathbf{x}}}\rangle)$ to measure the $|-\rangle$ state (differently put, the desired inner product corresponds to the values of the off-diagonal elements of the ancilla's density matrix [178] or the expectation value of the X operator [130]). One can reveal this probability with repeated runs of the algorithm and consequent measurements. If the probability of measuring the minus basis is smaller than 1/2, the classification is 1, while a probability $> 1/2$ leads to a classification of -1 (due to the

minus sign in the expression). This version of the swap test recovers the sign of the inner product, while the original swap routine only reveals $|\langle\psi_{\mathbf{u}}|\psi_{\tilde{\mathbf{x}}}\rangle|^2$. Rebentrost et al. furthermore introduce an interesting idea to implement a kernel trick for polynomial kernels in the quantum routine by using several copies of the respective quantum states (which will be further discussed in Chapter 7).

Zhao, Fitzsimons and Fitzsimons [130] adapt the routines for Gaussian processes (see Section 2.3.4.4). In Gaussian processes, the model distribution in Equation (2.38) is given by

$$p(y|\mathbf{x},\mathcal{D}) = \mathcal{N}\big[y|\underbrace{\boldsymbol{\kappa}^T\mathbf{K}^{-1}\mathbf{y}}_{\text{mean }\mu}\,,\ \underbrace{\kappa-\boldsymbol{\kappa}^T\mathbf{K}^{-1}\boldsymbol{\kappa}}_{\text{covariance }\delta}\big],$$

where $\kappa = \kappa(\mathbf{x},\mathbf{x}')$ is the kernel function applied to the new input, $\boldsymbol{\kappa}$ is a vector where the kernel function is applied to the new input and the respective training inputs, and $\mathbf{K}$ is the kernel Gram matrix for the training inputs with entries $\kappa(\mathbf{x}^m,\mathbf{x}^{m'})$ for $m,m' = 1...M$. Again the main computational task is the inversion of a $M \times M$ dimensional kernel matrix to compute the mean and variance at a new input $\tilde{\mathbf{x}}$. In order to compute these values, Zhao et al. essentially apply the HHL routine to prepare a quantum state representing $\mathbf{K}^{-1}\mathbf{y}$. For the inner product with $\boldsymbol{\kappa}^T$ they propose a similar trick to the one in Rebentrost et al. . Zhao et al. remark that there are indeed settings in which the conditions for their quantum algorithm are fulfilled, namely that the data as well as the kernel matrix is sparse and the kernel matrix is well conditioned, which might require some preprocessing as well as some approximations.

6.1.4 Further applications

The strategies developed in the above papers have been used in other contexts. Kerendis and Prakash [180] recognise that so called recommendation systems are an excellent case for applications. Recommendation systems are based on matrices R that store a rating of M users for N different items (an illustrative example is Netflix user rating for movies). The learning task is to predict the unknown rating of a user i for a specific item m, or the value R_{ij} of the matrix. A common observation is that these matrices are of low rank, which means that there are only a few prototypes of 'taste' and the data consequently exhibits high redundancy. Beyond pattern recognition or completion tasks there are other relevant proposals for linear algebra with quantum systems. One of them is data compression and preprocessing, for example to find the principal components of the data matrix via quantum principal component analysis [140]. Another suggestion is the statistical method of discriminant analysis [179].

In summary, there are a number of tools for linear algebra calculus with the amplitudes of a quantum system, and they have been used in various contexts where machine learning reduces to matrix inversion or singular value decomposition. Some important problems remain unresolved. Firstly, can we extend these methods to nonsparse and full-rank matrices but remain 'super-efficient'? Is there a way to prepare states that are nonsparse, but neither almost uniform polynomially in the number of qubits n? How can regularisation be included? Lastly, the proposals presented here all apply to machine learning methods that can be formulated as a linear system of equations. However, in principle the exponentially compact information encoding is open to other optimisation tasks as well. Is there a way to quantise iterative methods for nonconvex, nonquadratic optimisation problems? I will explore the last question further in Chapter 8.

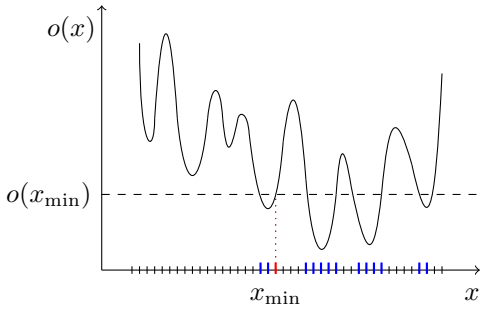


Figure 6.1: Illustration of Dürr and Høyer's optimisation algorithm. A quantum register in uniform superposition represents the inputs x in binary representation. In each iteration all states x with $o(x_{\min}) > o(x)$ (i.e., those with blue ticks) are marked by an oracle and their amplitudes are amplified . The register is measured and if the result x' fulfills $o(x_{\min}) > o(x')$, the current candidate for the minimum (red tick) gets replaced by x'.

6.2 Optimisation with Grover search

Only some optimisation problems have an elegant closed linear algebraic formulation. The others can always be formulated as search problems, i.e. to find the parameters θ that minimise the objective function $o(\theta)$. Exhaustive search is NP-hard, as scanning through 2^n combinations of n binary digits easily gets problematic. As far as we know today, quantum computing cannot break the 'NP-barrier' for unstructured search either, but only requires of the order $\sqrt{2^n}$ queries to an oracle, which may be interesting in some contexts of machine learning. Grover search is in this sense a 'plug-and-play' approach, allowing us to replace any search in an unstructured database with Grover search and harvest the quadratic speedup, provided the oracle implementation is given. A large number of proposals with varying intricacy made use of this idea already, and some of them shall be sketched in the following.

6.2.1 Finding closest neighbours

Dürr and Høyer [202] developed a quantum subroutine using Grover's algorithm to find the minimum of a function $o(x)$ with $x \in \{0,1\}^n$, and the routine can be used to find closest neighbours to a new input (see Figure 6.1). Let $\frac{1}{\sqrt{N}} \sum_j |j\rangle|x_{\text{curr}}\rangle$ be a superposition where we want to search through all possible $|j\rangle$. In each step an oracle marks all states $|j\rangle$ that encode an input x so that $o(x)$ is smaller than $o(x_{\text{curr}})$. To perform the comparison, x_{curr} is saved in an extra register. The marked amplitudes get amplified, and the $|j\rangle$ register measured. If the result x' is indeed smaller than x_{curr}, one replaces the current minimum by the newly found one, $x_{\text{curr}} = x'$. The routine gets repeated until the oracle runs empty, at which point x_{curr} is the desired minimum.

This routine has been applied to machine learning methods such as clustering [184] and k-nearest neighbour [181] (see Section 2.3.4.2). As an example I will briefly sketch the latter, namely Wiebe, Kapoor and Svore's quantum k-nearest neighbour routine [181]. The idea is straight forward: Starting with a dataset superposition in amplitude encoding (Equation 4.1), one uses a swap test (without the final measurement) to compute the inner products or Euclidean distances between the new input and the training vectors, uses amplitude estimation to write the distances into and extra register via basis encoding, and finally applies the Dürr-Høyer routine to find the state representing

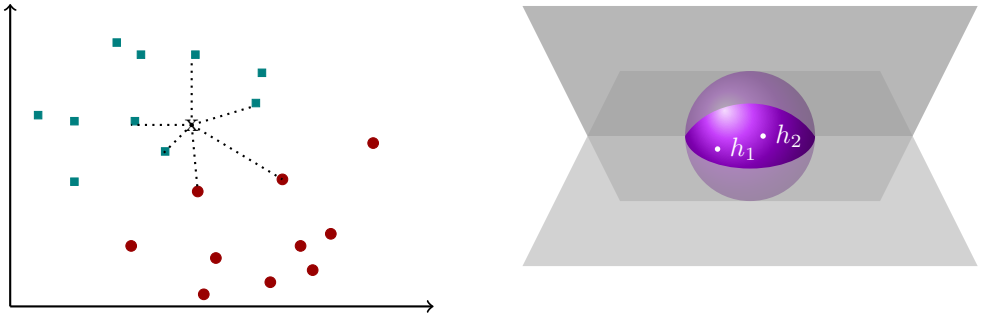


Figure 6.2: Illustrations of distance-based optimisation in machine learning. Left: A dataset with two different classes, purple circles and red rectangles. Clustering methods try to find the closest centroid to an input, while nearest neighbour methods try to find the closest data points. Right: In the dual representation, the separating hyperplanes h_1 and h_2 of a perceptron model can be translated into points on a hypersphere, while the training inputs identify a feasible region cut out by hyperplanes, in which the data is completely separated.

the smallest distance based on Grover search. This method seems rather cumbersome and has a number of preconditions such as state preparation (the authors assume sparse input vectors), but demonstrates how in principal Grover search can be used to identify the closest neighbour(s) of a training input. The method has recently also been proposed for a model of natural language processing [186].

Another interesting contribution has been made by Kapoor, Wiebe and Svore [147] with regards to the training of perceptrons. Perceptron models have the advantage that rigorous bounds for training are known. Using the standard learning algorithm outlined in Section 2.3.2.1 and for training vectors of unit length with classes separated by a margin γ, a trained perceptron makes at most $\frac{1}{\gamma^2}$ mistakes. Even in the best case, it has to check all M training inputs and therefore has a runtime at least linear in the number of training inputs. The authors use a dual representation of the hyperplanes that separate the data correctly. In this representation, the hyperplanes are depicted as points on a hypersphere while data points are represented as hyperplanes "that induce constraints on the feasible set of classifiers." [147] (see Figure 6.2 right). Applying Grover search on the 'hypersphere of hyperplanes' then induces a speedup $M \to \sqrt{M}$ and $\frac{1}{\gamma^2} \to \frac{1}{\sqrt{\gamma}}$.

In Chapter 9 I will show how to find close neighbours via interference without recurring on Grover search.

6.2.2 Quantum walks

Grover search is somewhat related to quantum walks, a quantum version of classical random walks in which the walker does not use a coin flip to decide which neighbouring node of a graph to visit next, but walks the graph in a quantum superposition (see for example [203]). A well-established quadratic speedup for the hitting time (at which there is a high probability to reach a predefined node in the graph) of quantum walks has been reported, and is based on interference effects of the amplitudes. Quantum walks are therefore another candidate to use in machine learning applications, specifically in replacement for classical random walks.

For example, Paparo and Martin-Delgado [182] approach Google's PageRank algorithm with a

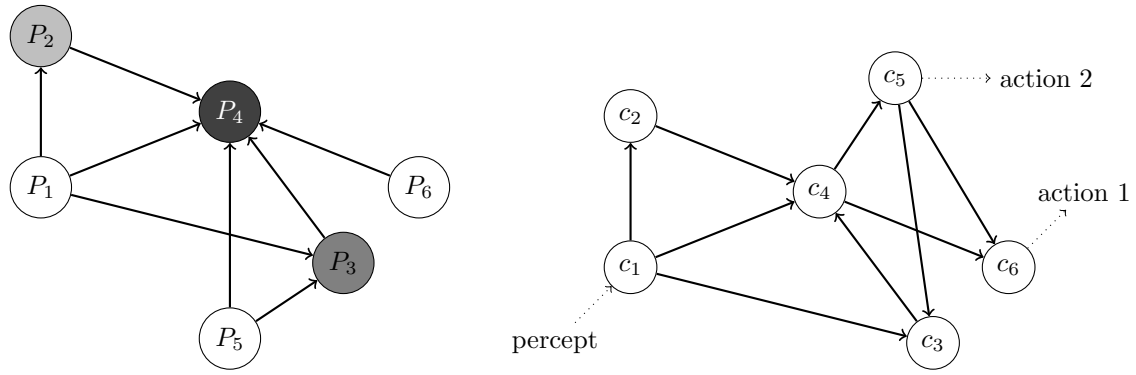


Figure 6.3: Examples of graph structures in machine learning applications. Left: In Google's PageRank algorithm, a graph of links (edges) between homepages (nodes) is created and the Markov process or random walk on the graph will show a high probability for nodes with lots of neighbours linking to them, such as node P_4. Quantum walks show evidence of revealing 'second hubs' such as node P_3. Right: In learning agents based on projective simulation, the agent performs a random walk in a graph of memorised 'clips'. Random walks can speed up the time the walk takes to lead to an action.

certain type of quantum walks (see Figure 6.3 left). PageRank assigns weights to homepages and was the core routine of the famous Google search routine. The ranking is based on a rule that gives high weights to sites that are linked by many other pages, while not linking to many other pages themselves. Homepages can therefore be understood as a graph where directed edges represent links between pages. A random walk (or Markov chain) on the graph will after many iterations lead to a state where nodes that have many directed edges leading to them have a high probability of being visited, while those with few connections or many outgoing edges get a much smaller 'weight'. Paparo and Martin-Delgado quantise the Markov chain associated with the PageRank algorithm and analyse the properties of the resulting quantum Markov chain via its spectral properties [182]. Simulations show that the quantum version weighs pages differently. In a follow up study [2] Paparo et al. find that the "quantum PageRank algorithm is capable of unveiling the structure of the graph to a finer degree", for example by giving more weight to "secondary hubs" of less importance than the most referenced pages. Another study by Loke et al. [183] confirms these findings and notes that the quantum version "resolves ranking degeneracy" of pages with a similar weight. This line of research is a beautiful example for the different quality of a quantum learning algorithm which have been discussed in Section 5.3.

A similar type of quantum walks have also been used in the context of active learning agents [2] based on the *projective simulation* model [204]. Projective simulation basically consist of a graph of interconnected 'memory clips', and upon a stimulus (new input) the agent performs a random walk on the graph until one of the memory clips is connected to an action (output) (see Figure 6.3 right). Replacing the random walk by a quantum walk can yield a quadratic speedup in the transversing of the graph from inputs to outputs, and claims another 'quadratic speedup for artificial intelligence'. For certain graph types, exponential speedups in hitting times have been established [205], and it remains to see if this can find application in machine learning as well.

There are many more contributions that apply Grover search to machine learning, but they follow similar rationales to the ones mentioned here.

6.2.3 Pattern completion

Associative memory refers to a memory retrieval mechanism in Hopfield neural networks (introduced in Section 2.3.2.3) in which the memorised patterns are stored in the ground state of an Ising-type energy function, and upon initialisation of the recurrent neural network the dynamics drive its state towards the closest memorised pattern. This mechanism can also be used for supervised pattern recognition, namely if the incompleteness refers to the output bit. Without the original connection to biological neural networks, the task can be formulated as a simple search problem to find the closest pattern to a new input $\tilde{x}$ in a database of memorised patterns $\mathbf{x}^1, ..., \mathbf{x}^M$, which is in fact a type of k-nearest neighbour with $k = 1$. (In the original proposals of associative memory the features are usually binary and the distance is the Hamming distance). Writing the memorised pattern dataset into a quantum state in basis encoding,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |\mathbf{x}^m\rangle$$

opens the avenue to apply Grover search. For example, one can mark all states of a certain Hamming distance to the new input and increase their amplitude. One problem is that Grover iterations do not preserve zero amplitudes and therefore create 'artificial' training inputs in the database which can disturb the algorithm severely.

Ventura and Martinez [116] therefore introduce a simple adaptation to the amplitude amplification routine. After the desired state is marked and the Grover operator is applied for the first time reflecting all states about the mean, a new step is inserted which marks *all* states in the database superposition. The effect gives all states but the marked one the same phase and absolute value, so that the search can continue as if starting in a uniform superposition of all possible states. Another solution for the pattern completion problem is presented by Salman et al. [185] who use two oracles, one to mark the states in the database and one to mark the solutions of the problem. Applying a variation of Grover's algorithm that finds the intersection of marked states of two different oracles then increases the amplitudes of the desired basis states. If the problem allows for more than one solution, the number of solutions have to be determined by quantum counting routines [94] in order to estimate the optimal number of repetitions for Grover search.

6.3 Sampling from quantum states

The idea of sampling from quantum states is rather straight forward. One prepares a quantum system in a state that corresponds to a *qsample* $\sum_{i=0}^{N-1} \sqrt{p_i}|i\rangle$ of a desired distribution $p = \{p_0...p_{N-1}\}$. A measurement in the computational basis effectively samples from p. The qsample can either be prepared by a circuit (as in quantum rejection sampling), or it is a generic distribution that occurs in a specific quantum system, such as a quantum annealing device. Both ideas have been investigated for quantum machine learning.

6.3.1 Boltzmann sampling from quantum annealing devices

In the training of Boltzmann machines introduced in Section 2.3.2.4 the central computational problem stemming from maximum log likelihood optimisation is to calculate the 'expectation

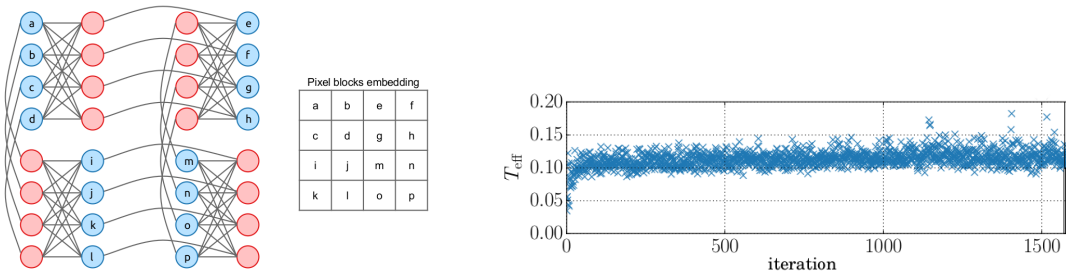


Figure 6.4: Challenges when sampling from the D-Wave device. Left: The information (here a 4×4 matrix) has to be encoded in a Chimera graph structure that describes how qubits are interconnected in the hardware. The device has an effective temperature, here estimated for iterative sampling by Ref. [206]. Both graphics are taken from [206].

value'

$$\langle s_i h_j \rangle_{\text{model}} = \sum_{s,h} \frac{e^{-E(s,h)}}{Z} s_i h_j$$

of the correlation between the hidden units h_j and the visible units s_i over the Boltzmann distribution $\frac{e^{-E(s,h)}}{Z}$ (see Equation (2.25)). The Energy function is of an Ising or spin-glass type,

$$E(s,h) = -\sum_{ij} w_{ij}\ s_i h_j - \sum_{jj'} w_{jj'}\ h_j h_{j'} - \sum_{ii'} w_{ii'}\ s_i s_{i'},$$

where in the common case of restricted Boltzmann machines, the 'interaction strengths' $w_{jj'}, w_{ii'}$ are zero. Instead of calculating the exact expectation value, it is usually sufficient to draw samples from the Boltzmann distribution, which is not much less difficult due to the partition function Z summing over an exponential number of terms. Some researchers proposed to use the natural distributions generated by quantum annealers such as the D-Wave as approximations to the Boltzmann distributions [63]. Strictly speaking these would be quantum Boltzmann distributions where the (Heisenberg) energy function includes a transverse field term for the spins, but under certain conditions quantum dynamics might not play a major role. Experiments on the D-Wave recently showed that distributions prepared by the device can in fact significantly deviate from a classical Boltzmann distribution, and the deviations are difficult to model due to out-of-equilibrium effects of fast annealing schedules [187]. However, the distributions obtained still seem to work very well for the training of Boltzmann machines, and improvements in the number of training steps compared to contrastive divergence have been reported from a number of groups both for experimental applications and numerical simulations [188, 187, 206, 189, 207]. As an application, samples from the D-Wave device have been shown to successfully reconstruct and generate handwritten digits [207, 188]. A great advantage compared to contrastive divergence is also that fully connected models can be used.

While conceptually, the idea of using physical hardware to generically 'simulate' a distribution seems very fruitful, the details of the implementation with the relatively new hardware are at this stage rather challenging [207]. For example, researchers report Gaussian noise pertubating the parameters fed into the system by an unknown function (and with typical deviations of the order of 0.03 [187]), which prohibits precise calibration. A constant challenge is also the translation of the coupling of visible and hidden units (or even an all-to-all coupling for unrestricted Boltzmann

machines) into the sparse architecture of the connections between qubits in the D-Wave quantum annealer (see Figure 6.4). Fully connected models need to be encoded in a clever way to suit the typical chimera graph structure. Another serious problem is the finite 'temperature' of the device. Effectively, the distribution implemented by the machine contains a temperature parameter β in $\frac{e^{-\beta E(s,h)}}{Z}$, which is unknown and fluctuates during the annealing process. A promising strategy was recently proposed to estimate this internal temperature from the samples drawn [206]. Solving these problems is a crucial step towards realistic quantum implementations of machine learning algorithms and and are likely to become important for small-scale universal quantum devices in the near future.

6.3.2 Boltzmann sampling from quantum states

While quantum annealers seem to generically prepare distributions that can be used for sampling in the training of Boltzmann machines, one can also think of preparing a qsample of a Boltzmann distribution on a universal quantum computer, and formulated in the gate model. This strategy has been investigated by Wiebe et al. [208] based on quantum rejection sampling suggested by Ozols and co-workers [98] (see Section 3.4.4). The basic idea is to first prepare a qsample of a mean field approximation of the desired distribution (which has to be computed classically beforehand) and then use postselective amplitude updates to refine the distribution. The advantage of these two steps is that the mean-field approximation factorises in the qubits and can therefore be prepared by turning each qubit successively. At the same time, the postselective amplitude update has much more chances to be successful if applied to a distribution that is already close to the final Boltzmann distribution.

To go into a bit more detail, the mean-field distribution $q(s,h)$ is defined as the product distribution that minimises the Kullback-Leibler divergence to the desired Boltzmann distribution $p(s,h)$ (which is a common measure of distance between distributions). Summarising the $N+1$ visible units $\{s_i\}$ and the J hidden units $\{h_j\}$ for now as variables $v_1...v_{N+1+J}$, the approximate distribution can be written as

$$q(v) = q_1 \cdot q_2 \cdot ... \cdot q_{N+1+J}, \;\; q_i = \begin{cases} \mu_i \text{ if } v_i = 1 \\ 1-\mu_i \text{ else} \end{cases} \quad , \quad 0 \leq \mu_i \leq 1 \;\; , i = 1, ..., N+1+J.$$

The mean field parameters μ_i have to be obtained in a classical calculation. To prepare a qsample of the form

$$\sum_{s,h} \sqrt{q(s,h)} |s,h\rangle,$$

where $|s,h\rangle = |s_1...s_{N+1}, h_1...h_J\rangle$ abbreviates the state of the network in basis encoding, one simply has to rotate qubit i by $\sqrt{\mu_i}$ in z-direction. We assume furthermore that a real constant $k \geq 1$ is known such that $p(s,h) \leq kq(s,h)$.

The second step requires us to add an extra register and load another distribution in basis encoding into it to obtain

$$\sum_{s,h} \sqrt{q(s,h)} |s,h\rangle |\bar{p}(s,h)\rangle.$$

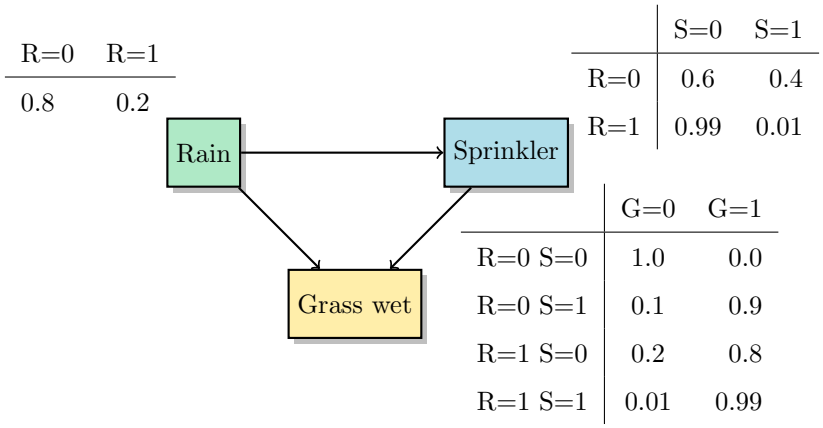


Figure 6.5: The Bayesian network example from Section 2.3.3.1, which is used to demonstrate the preparation of a qsample to perform quantum inference on Bayesian networks. The 'R' node stands for rain (No = 0, Yes = 1), 'S' refers to the sprinkler being on or off and 'G' indicates if the grass is wet or not.

This second distribution is constructed in such a way that $q(s,h)\bar{p}(s,h)$ is proportional to $p(s,h)$. A postselective amplitude update then leads to the target state with success probability larger than $1/k$. If the approximation of $q(s,h)$ was exact, $k = 1$ and the postselection succeeds with certainty. The larger the divergence between $q(s,h)$ and $p(s,h)$, the smaller the success of the conditional measurement (as in the case of classical rejection sampling). The authors therefore admit that the method cannot be effective and accurate for every task, but works in cases where the mean-field approximation is successful. Simulations show that this method performs rather well, and the work led to a classical version of the algorithm [209].

6.3.3 Sampling from quantum states encoding Bayesian nets

The quantum rejection sampling routine has been applied to Bayesian nets (see Section 2.3.3.1) in a beautiful paper by Low, Yoder and Chuang [148]. To recapitulate, Bayesian nets with units $s_1 ... s_{N+1}$ are described by a probability distribution of the form

$$p(s_1, ..., s_{N+1}) = \prod_{i=1}^{N+1} p(s_i|\pi_i),$$

where π_i is the set of parent nodes to $s_i \in \{0,1\}$. To prepare a qsample of this probability distribution, take a quantum state of $N+1$ qubits corresponding to the nodes, and rotate qubit q_i from $|0\rangle$ to $|1\rangle$ by a value $p(s_i|\pi_i)$ conditioned on the state of the parents.

In the example Bayesian net from Section 2.3.3.1 reprinted in Figure 6.5, one would require a register of three qubits $|0_R 0_S 0_G\rangle$ to represent the three variables. The only node without a parent is R, and the rotation is therefore unconditional,

$$|0_R 0_S 0_G\rangle \rightarrow (\sqrt{0.8}|0_R\rangle + \sqrt{0.2}|1_R\rangle)|0_S 0_G\rangle.$$

Now rotate every successive node of the belief network conditioned on the state of its parents. This is possible because of the acyclic structure of the graph. For example, the second qubit will be

rotated by $\sqrt{0.4}$ or $\sqrt{0.01}$ controlled by the first qubit being in $|0\rangle$ or $|1\rangle$,

$$|R0_S0_G\rangle \rightarrow \left[\sqrt{0.8}\,|0\rangle(\sqrt{0.6}\,|0\rangle + \sqrt{0.4}\,|1\rangle) + \sqrt{0.2}\,|1\rangle(\sqrt{0.99}\,|0\rangle + \sqrt{0.01}\,|1\rangle)\right]|0_G\rangle$$

Rotating the last qubit $|0_G\rangle$ requires four rotation gates each controlled by two qubits to obtain

$$|RSG\rangle = \sqrt{0.48}\,|100\rangle + \sqrt{0.032}\,|101\rangle + \sqrt{0.00002}\,|110\rangle + \sqrt{0.00198}\,|111\rangle.$$

Obviously, for each qubit one needs $2^{|\pi_i|}$ rotations ($|\pi_i|$ is the number of s_i's parents). The resources for state preparation therefore grow with $\mathcal{O}((N+1)2^{|\pi_i|_{max}})$, where $|\pi_i|_{max}$ is the highest number of parents eny node has. We can therefore prepare quantum states for sparsely connected models well.

In classical Bayesian nets, inference relates to the task of calculating the probability of some unit s_k being in state 0 or 1 given the evidence of the states of all other units, $p(s_k|e)$. Remember that this can be used for a supervised pattern classification task if the evidence is interpreted as a new input and s_k is the desired output. One can draw samples from $p(s_k|e)$ by first sampling from $p(s_1, ..., s_{N+1})$ (by following the graph structure and sample each variable given its parents' state), and consequently rejecting samples that do not correspond to the evidence. The successful samples are then effectively drawn from the desired distribution. The chance of accepting a sample is given by the probability of the evidence to occur, $p(e)$. The average number of original samples needed to get one accepted sample is in $\mathcal{O}(1/p(e))$.

Sampling from the quantum state as in the quantum rejection sampling algorithm (Section 3.4.4) can improve this factor quadratically. Formally, one can divide the qsample[1] into the part of the superposition containing the evidence, and the rest,

$$\sqrt{p(e)}\,|s_k, e\rangle + \sqrt{1-p(e)}\,|s_k, \bar{e}\rangle.$$

Applying amplitude amplification to this state linearly increases the probability of the branch containing the evidence. The state preparation operator has to be applied $\mathcal{O}(1/\sqrt{p(e)})$ times. In summary, the quantum sampling method for Bayesian inference runs in $\mathcal{O}(n2^{|\pi_i|_{\max}}\frac{1}{\sqrt{p(e)}})$.

I will use quantum sampling methods for ensemble methods in Chapter 11.

6.4 Optimisation with ground states of Hamiltonians

The fourth type of quantum machine learning algorithms uses quantum systems as analogue computational devices to solve problems such as combinatorial optimisation. The solution of the optimisation problem is encoded into the ground state of a quantum system, the quantum state of n qubits with the lowest energy. For this, the dataset typically defines the Hamiltonian (and thereby the dynamics) of the system, which can be understood as a type of *dynamic encoding*. This approach is closely related to the adiabatic model of quantum computation introduced in Section 3.2.3 and its heuristic of quantum annealing. Note that although some sampling

[1] An interesting point made in [148] is that a quantum probability density function for the distribution $p(s)$ should in general be defined as a qsample that fulfils the property of *q-stochasticity*, which means that for every stochastic matric $T : p(s) \rightarrow p'(s)$, there is a unitary that maps $|\psi_p\rangle \rightarrow |\psi_{p'}\rangle$.

approaches discussed above also use quantum annealing, their purpose is to draw samples from a probability distribution, so to speak the by-product of the dynamic process, and not to retrieve the ground state.

6.4.1 Rewriting problems as quadratic unconstrained binary optimisation

Most physical systems used for quantum annealing consist of n qubits with programmable interaction strengths in an Ising-type model. For example, the D-Wave device anneals objective functions in the format of *quadratic unconstrained binary optimisation*. The ground state or solution is the binary sequence $x_1...x_n$ that minimises the 'energy function'

$$\sum_{i\leq j=1}^{n} w_{ij}x_ix_j., \tag{6.1}$$

with the coefficients or weights w_{ij}. A central task for this approach of quantum machine learning is therefore to cast a learning task into a suitable optimisation problem.

QBoost developed by researchers at Google and D-Wave [210, 149, 105] is a beautiful illustration of how casting a classification problem into the format amenable for quantum annealing devices can also lead to new methods for machine learning. In such cases the quantum translation of the algorithm has to be compared with the corresponding novel classical machine learning model. (I will consider such a case in my own work in Chapter 11.) The basic machine learning model is an ensemble of K classifiers $h_k(\mathbf{x})$, $k = 1, ..., K$, that are combined by a weighted sum of the form $f(\mathbf{x}; \mathbf{w}) = \text{sgn}(\sum_{k=1}^{K} \mathbf{w}^T h_k(\mathbf{x}))$ with $\mathbf{x} \in \mathbb{R}^N$ and $y \in \{-1, 1\}$ (see Figure 6.6). Training minimises a least-squares loss function together with a L_0 regularisation term to prevent overfitting

$$\frac{1}{M}\sum_{m=1}^{M}(f(\mathbf{x}^m; \mathbf{w}) - y^m)^2 + \lambda|\mathbf{w}|_0,$$

where $|\ \ |_0$ counts the number of nonzero parameters (and is hence a sparsity-inducing norm). The hyper-parameter λ tunes the strength of regularisation. The problem is already in the form of quadratic unconstrained binary optimisation as can be seen by evaluating the square brackets:

$$\sum_{k,k'=1}^{K} w_k w_{k'} \left(\sum_{m=1}^{M} h_k(x^m)h_{k'}(x^m)\right) + \sum_{k=1}^{K} w_k \left(\lambda - 2h_k(x^m)y^m\right).$$

The known terms $\sum_m h_k(x^m)h_{k'}(x^m)$ and $\lambda - 2h_k(x^m)y^m$ serve as the interaction and field strengths of the Ising model, while the weights take the role of the x_i, x_j from Equation (6.1), which is sometimes called an *inverse Ising model*. However, the formulation requires that the weights are binary variables w_i. One can replace them by binary strings with a little more modelling effort, but current quantum hardware limits us to a very low bit depth τ if $w_i \in \{0, 1\}^\tau$. Estimations show that low-bit representations of parameters can still represent a sufficiently rich space of decision boundaries, or more precisely that "the required bit depth for weight variables only grows logarithmically with the ratio of the number of training examples to the number of features" [210]. This is consistent with some successful application of binary weights to deep

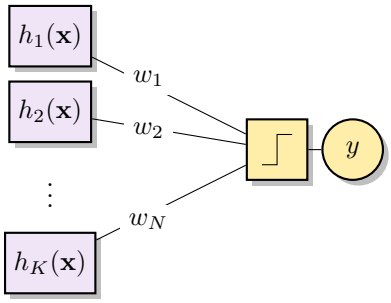


Figure 6.6: Illustration of QBoost. Several classifiers $h_k(\mathbf{x})$ are combined by a perceptron-like gating network that builds the weighed sum of the indiviual predictions and applies a step activation function on the result to retrieve the combined classification decision y.

neural networks [211].

Neven and his co-workers claim that compared to the well-known ensemble method AdaBoost, the structure of the QBoost problem shows various advantages even if executed on a classical computer (and with a predefined set of weak classifiers $h_i(x)$): It produces a strong classifier that in some cases is able to beat classical AdaBoost through a lower generalization error, employing fewer weak classifiers and requiring fewer boosting iterations. These advantages are suspected to be largely due to an intrinsic regularisation by the low-bit depth binary representation of the weights and the good performance of the least-square objective function (as opposed to AdaBoost's stepwise weighing of the training vectors). Denchev et al. [197] add onto this work by introducing a special loss function (*q-loss*) for perceptron models of the form used for QBoost. This loss function can also be formulated as a quadratic unconstrained optimisation problem, but promises better robustness against large outliers than the square loss. Less optimistic results are obtained by Dulny and Kim [193], who find that in a realistic setting using a *Kaggle* competition dataset, classical state-of-the-art methods perform better than QBoost. However, they attribute these findings to the reformulation of the optimisation problem which tries to meet the hardware restrictions, rather than the quantum annealing technique as such. However, QBoost opens avenues for a lot of exciting research.

Machine learning problems that lend themselves especially well for combinatorial optimisation are structure learning problems for graphs, for example of Bayesian nets [196]. Each edge of the fully connected graph corresponds to a qubit, and under the assumption that no node has more than $|\pi|_{\max}$ parents a 'score Hamitonian' can be defined that assigns an energy value to every permitted graph architecture/ bit string. Other examples of casting machine learning problems into quadratic unconstrained optimisation problems are provided by Neven, Rose and Macready who look at image matching (recognising that two images show the same content but in different light conditions or camera perspectives)[212], as well as Pudenz and Lidar [194] who give an extensive example for software verification and validation.

A special problem that naturally comes in the shape of a quadratic unconstrained optimisation problem is the memory recall of a Hopfield neural network (see Section 2.3.2.3). In Equation 6.1 the weights w_{ij} are determined by a learning rule such as the Hebb rule, and define the solution to the corresponding minimisation problem. The x_i, x_j are bits of the new input pattern with

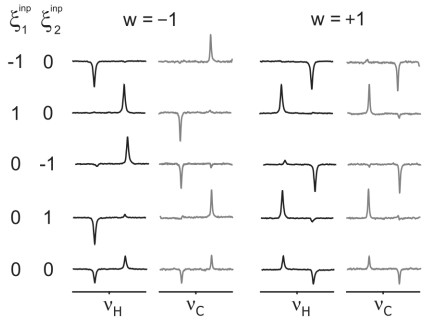


Figure 6.7: Associated memory recall with atomic spectra of ^{1}H and ^{13}C. The spectra encode the results obtained for the inputs $\chi_1^{\text{inp}}, \chi_2^{\text{inp}}$ and under the weight settings $w = \pm 1$. Image reprinted from [191].

binary features. The updating rule of Hopfield neural networks is based on a perceptron activation function which drives the state of the system upon initialisation to a close ground state. This process is called an associative memory recall. A straight forward method to translate this into quantum algorithms is to replace the recall with adiabatic quantum computing or annealing.

Work has also been done beyond quadratic unconstrained optimisation. For example, an early contribution was made by Horn and Gottlieb [199]. Their idea is to define a potential V for a Hamiltonian (which is traditionally decomposed into a kinetic and potential energy, $H = T + V$) such that a wave function with high probabilities in regions of high data density is an eigenvalue to the Hamiltonian. The wave function serves as a kernel density estimator function introduced in Section 2.3.4.1. This idea has been recently used to find the optimal radial basis activation functions in neural networks [200].

6.4.2 Finding the ground state of the quantum system

After the problem has been formulated so that it can be implemented in a quantum information processing device, techniques of adiabatic quantum computing or quantum annealing can be applied to drive the system into its ground state. QBoost, Bayesian network structure learning and associative memory recall have all been tested on the D-Wave machine with the somewhat ambiguous results mentioned above. Other studies paint a similar picture, such as Santra et al. [166] who simulate a Hopfield neural network model with a quantum Ising (more precisely, a Heisenberg) Hamiltonian and implement the associative memory recall on a D-Wave machine, or Sato et al. [195] who replace simulated annealing for variational Bayes (i.e., to compute intractable integrals in Bayesian inference) with quantum annealing to escape local minima in the optimisation landscape. The diverse range of outcomes is no surprise at this stage, at which comparisons of classical and quantum annealing are not yet fully conclusive (see for example [77, 143]).

Other physical implementations than quantum annealers have been used to investigate the power of quantum optimisation in the machine learning context. For example, Neigovzen [191] demonstrate that associative memory recall can be executed with a nuclear magnetic resonance setup. They encode two binary values into the spectrum of ^{1}H and ^{13}C nuclear spins. Roughly said, for each qubit a positive peak at the left region of the spectrum indicates a 1 while

a negative peak in the right part of the spectrum indicates a -1. No signal leaves the input blank, encoding a 0. Setting a weight parameter[2] to $w = 1$ stores the two memory patterns $(-1, 1), (1, -1)$ into the time signal of the NMR system, while the configuration $w = -1$ stores the patterns $(-1, -1), (1, 1)$ (see Figure 6.7). Inputs with blanks at different positions lead to the desired memory recall. For example, for $w = 1$ input $(-1, 0)$ recalls the memory pattern $(-1, 1)$. This is a minimal example of a quantum associative memory. Interesting enough, the quantum model can lead to ground states that are superpositions of patterns, a phenomenon that should be investigated further and needs to be carefully compared with stochastic methods of classical models.

As a conclusion to this short literature review, most quantum algorithms for supervised pattern recognition follow the same strategy. They identify a difficult computational task in one of the well-established machine learning methods introduced in Chapter 2 and outsource it to a quantum computer. The computational tasks are mostly related to optimisation. Examples were matrix inversion, finding the smallest distance, traversing a graph with a random walk, solving a combinatorial optimisation problem, or sampling from a given distribution.

An important thought arises and will be picked up again in the conclusion to this thesis. The models and optimisation problems formulated in classical machine learning are tailor-made for classical computers. Figuratively spoken, the nails are shaped so that they can be worked on by a hammer. If we exchange the hammer by a tool from the quantum information processing toolbox, should we not rather rethink the shape of the nail as well? Instead of mimicking classical methods, hopefully with a proven speed-up, should we not come up with alternatives to least-squares optimisation, linear models and contrastive convergence? The approach of Hartmut Neven shows this beautifully: He chooses an optimisation problem that is not necessarily common in the machine learning community for large-scale classifier, but easily fits into the framework of quantum annealing with the D-Wave machine. This shifted perspective would require a lot of knowledge about machine learning, but would certainly lead to innovation in classical learning beyond speeding up well-established classical algorithms.

[2]Seddiqi et al. study the method of adiabatic quantum computing for associative memory recall in a theoretical framework and find that the way the accuracy of the recall depends on different parameters such as the noise depends heavily on the learning rule used [192].

Part III

Developing new quantum machine learning algorithms

The last part of this thesis is dedicated to the heart piece of my own research, which is the development of different quantum machine learning algorithms. Each chapter contains published material, with the reference and my contribution mentioned at the beginning of the chapter. In some cases this thesis extends the work, while recent manuscripts developed the ideas further.

Chapters 7 and 8 are more technical investigations into the first type of algorithm identified in the literature review. Based on the concept of amplitude encoding, they combine and extend different quantum routines to solve convex and non-convex optimisation problems. While 7 looks at the well-known method of least-squares linear regression, 8 more generally considers a quantum implementation of gradient descent methods for a particular case.

Chapter 9 tries to use interference to compute the distance measure of a kernelized classifier for both amplitude and basis encoded information and analyses the performance of the resulting model. Chapter 10 shows a framework to invesigate neural network models and presents a quantum circuit that implements a nonlinear step-activation function as a building block of a potential neural network. This building block could also be used as a base model for the quantum ensemble classifier in Chapter 11, which explores how quantum superposition can construct exponentially large committees of experts.

Chapter 7

Quantum linear regression algorithm

Sections 7.1 and 7.2 of this chapter are based on Ref [213]: Schuld, Sinayskiy and Petruccione (2016) Prediction by linear regression on a quantum computer, Physical Review A, 94 (2): 022342. I was responsible for the idea, development, analysis and write-up of the content of the publication.

As outlined in Section 2.3.1.1, regression in the context of supervised pattern recognition refers to the task of fitting a function $f : \mathbb{R}^N \to \mathbb{R}$ to data points in $\mathcal{X} \times \mathcal{Y} \in \mathbb{R}^N \times \mathbb{R}$ in order to capture their fundamental trend, and to use it to predict the output $\tilde{y} \in \mathcal{Y}$ for a new input $\tilde{x} \in \mathcal{X}$. Data fitting has been investigated by statisticians for over a century (see [214] for a translation of Gauss' works from 1820). One of the most popular methods of optimising the fit of a linear function in statistics is least squares optimisation, in which the collective squared distance between model predictions $f(x)$ and data points is minimised. Least-squares optimisation is a convex quadratic optimisation problem and the solution can be written in a closed form equation. Calculating the solution typically requires the inversion of a matrix created from the data (see Section 2.3.1.1).

Machine learning has adopted the method of linear regression to train a linear classifier and simply added the step of classifying a new input with the fitted funtion for prediction. However, in contrast to typical statistics problems one usually deals with highly complex (and by no means linear) relationships between inputs and outputs, and nonlinear feature maps of the data into higher-dimensional input spaces become important tools. Furthermore, regularisation techniques enforce the solution (i.e., the parameters of the fit) to be sparse or low-valued. While in statistics, this allows simple models to be inferred, in the machine learning context it also prevents overfitting in the feature space.

Due to its dependency on matrix inversion, linear regression appears to be an excellent candidate to apply the techniques of quantum matrix inversion developed by Harrow, Hassidim and Lloyd [90]. In the literature review in Section 6.1 the quantum algorithm by Wiebe, Braun and Lloyd [128] for statistical data fitting with sparse datasets has been introduced. The output to the algorithm is a quantum state whose amplitudes correspond to the optimal fit parameters. The authors conclude that a read-out of all parameters would destroy the logarithmic dependency on

the input size and proceed to analyse estimators for the quality of a fit. I consequently argued that an open problem is the application of the quantum matrix inversion technique in the machine learning context, i.e. including the prediction step. Also, it would be favourable if the quantum machine learning algorithm would take nonsparse inputs, as only very specific datasets consist only of a few nonzero features.

I address this problem by introducing a quantum machine learning algorithm for prediction based on linear regression. As many other quantum algorithms for linear algebra calculations it is essentially a combination of different tools presented before. In this case the goal is to implement a singular value decomposition of the data matrix, and to use it for prediction. After state preparation, the routine is logarithmic in the input dimension and dataset size if the data matrix can be represented by a low-rank decomposition, which effectively means that it is dominated by a few large eigenvalues. The reason for this lies in the cost for a high accuracy of the subroutines, which would be needed to resolve small eigenvalues to a sufficient precision.

In order to keep things as simple as possible, the quantum machine learning algorithm will be derived in its most basic form, i.e. without a feature map and regularisation term. The last two sections will then discuss ideas of implementing regularisation and feature maps to harvest the full power of linear regression for pattern recognition. This is an extension to the published article, and has not been considered in the related work by Wiebe, Braun and Lloyd.

7.1 Formulation of the classical optimisation problem

The goal of the quantum machine learning algorithm is to reproduce the classical prediction result $\tilde{y}$. This section lays the theoretical foundations and derives the prediction result based on a singular value decomposition of the data matrix. The final expression to be reproduced by the quantum algorithm can be found in the last Equation (7.4).

Remember that a linear model is a map

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T \mathbf{w}, \tag{7.1}$$

where $\mathbf{w} = (w_1, ..., w_N)^T \in \mathbb{R}^N$ is a vector of parameters that has to be learnt from the data. The bias term w_0 can be included in the parameter vector $\mathbf{w}$ by extending the feature vectors $\mathbf{x}$ with an entry $x_0 = 1$. The least squares method defines the optimal parameter vector as the one that minimises the squared distance between $f(\mathbf{x}^{(m)}, \mathbf{w})$ and the target output $y^{(m)}$ for each datapoint,

$$o(\mathbf{w}) = \sum_{m=1}^{M} (f(\mathbf{x}^{(m)}, \mathbf{w}) - y^{(m)})^2,$$

which is the objective function of the optimisation problem. Regularisation terms are usually added to the right hand side and pose constraints to the optimal solution, enforcing either sparsity or a small length of the parameter vector.

For a more compact notation, one can write the training inputs into the rows of a 'data matrix'

$\mathbf{X} = (\mathbf{x}^{(1)}, ..., \mathbf{x}^{(M)})^T$, summarise the target outputs as the target vector $\mathbf{y} = (y^{(1)}, ..., y^{(M)})^T$, and write the least squares optimisation problem as

$$\hat{\mathbf{w}} = \min_{\mathbf{w}} |\mathbf{X}\mathbf{w} - \mathbf{y}|^2.$$

If the parameter vector would define a linear function that fit the data perfectly well[1], we would have $\mathbf{X}\mathbf{w} = \mathbf{y}$ and consequently $\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$ for invertible $\mathbf{X}$. The solution to the problem 'with noise' looks in fact very similar: Writing $|\mathbf{X}\mathbf{w} - \mathbf{y}|^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^\dagger(\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}$ and building the derivative $\partial_{\mathbf{w}} o(\mathbf{w})$ results in $-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w}$, which is zero at the minimum. The solution to the least squares optimisation problem for a linear regression model is therefore

$$\hat{\mathbf{w}} = \mathbf{X}^+\mathbf{y} \tag{7.2}$$

with $\mathbf{X}^+ = (\mathbf{X}^\dagger\mathbf{X})^{-1}\mathbf{X}^\dagger$. The matrix $\mathbf{X}^+$ fulfills the Moore-Penrose conditions for a *pseudoinverse*, a generalisation of the inverse for nonsquare or square singular matrices.

While Wiebe et al [128] use this derivation and implement a quantum operator with the same effect as $\mathbf{X}^+$ on a quantum state $|\psi_{\mathbf{y}}\rangle$, I will use the reduced singular value decomposition (see Box 7.1.1). The singular value decomposition of the data matrix is given by $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\dagger$. In this expression $\mathbf{U}$ is a $M \times R$ matrix which contains the left singular vectors $\mathbf{u}_r$ as columns, while the $J \times R$ dimensional $\mathbf{V}$ contains the right singular vectors $\mathbf{v}_r$. In the *reduced* version of the singular value decomposition, the diagonal matrix $\mathbf{\Sigma}$ only contains nonzero singular values $\sigma_1, \sigma_2, ..., \sigma_R > 0$ (for more details consult Box 7.1.1).

Inserting the singular value decomposition into Equation 7.2 we get

$$\begin{aligned}
\mathbf{X}^+ &= (\mathbf{X}^\dagger\mathbf{X})^{-1}\mathbf{X}^\dagger \\
&= (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\dagger\ \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\dagger)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\dagger \\
&= \mathbf{V}\mathbf{\Sigma}^{-2}\mathbf{V}^\dagger\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\dagger \\
&= \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\dagger.
\end{aligned}$$

Using the singular vector formulation this can be written as:

$$\mathbf{X}^+ = \sum_{r=1}^{R} \frac{1}{\sigma_r}\mathbf{v}_r\mathbf{u}_r^T,$$

Since quantum theory prominently features eigenvalues rather than singular values, it is important to point out that there are close relationships between the two, which I will exploit in the next Section. The singular values σ_r of $\mathbf{X}$ are at the same time the square roots of the nonzero eigenvalues λ_r of $\mathbf{X}^\dagger\mathbf{X}$ *and* $\mathbf{X}\mathbf{X}^\dagger$, or

$$\sigma_r = \sqrt{\lambda_r}.$$

The left and right singular vectors are the eigenvectors of $\mathbf{X}^\dagger\mathbf{X}$ and $\mathbf{X}\mathbf{X}^\dagger$ respectively.

[1] In a statistics setting this means that there is no noise present. In machine learning, the notion of noise is not as prominent in this case, as we do not expect that the data was in fact produced by a linear law in the first place.

Box 7.1.1: Singular Value Decomposition

A singular value decomposition is the generalisation of a eigendecomposition $\mathbf{A} = \mathbf{SDS}^\dagger$ which expresses a matrix $\mathbf{A}$ by two matrices, $\mathbf{D}$ containing the eigenvalues λ on the diagonal and $\mathbf{S}$ containing the eigenvectors $\mathbf{s}$ as columns. In other words, one uses the eigenvalue equations $\mathbf{As} = \lambda\mathbf{s}$ to decompose a matrix. Since $\mathbf{S}$ is a unitary matrix (or for real numbers, orthogonal), the inverse of $\mathbf{A}$ can now easily computed by inverting the eigenvalues on the diagonal of $\mathbf{D}$ and taking the complex conjugate of $\mathbf{S}$. For singular matrices that have no inverse, this is obviously not possible and no eigenvalue decomposition can be found. However, one can still formulate a *singular value decomposition* based on the singular value equations $\mathbf{Av} = \sigma\mathbf{u}$. The vector $\mathbf{v}$ [$\mathbf{u}$] is the right [left] *singular vector* to the *singular value* σ. This gives rise to a matrix decomposition of the form $\mathbf{A} = \mathbf{U\Sigma V}^\dagger$, where $\mathbf{U}, \mathbf{V}$ have the corresponding singular vectors as columns and $\mathbf{\Sigma}$ is a diagonal matrix of appropriate dimension containing the singular vectors. The inverse of $\mathbf{A}$ is likewise calculated by inverting the singular values on the diagonal and taking the complex conjugate of $\mathbf{U}, \mathbf{V}$. The singular value decomposition can always be found [215].

With this formulation of the pseudoinverse, the least squares solution reads

$$\hat{\mathbf{w}} = \sum_{r=1}^{R} \frac{1}{\sigma_r} \mathbf{u}_r^T \mathbf{y}\ \mathbf{v}_r, \tag{7.3}$$

and plugging this into Eq. (7.1) defines the output $\tilde{y}$ for a new input $\tilde{\mathbf{x}}$ as

$$\tilde{y} = \sum_{r=1}^{R} \frac{1}{\sigma_r} \tilde{\mathbf{x}}^T \mathbf{v}_r \mathbf{u}_r^T \mathbf{y}. \tag{7.4}$$

This result states that once the singular values σ_r and singular vectors $\mathbf{v}_r, \mathbf{u}_r$ of the data matrix are found we can use them to calculate the new output. The singular value decomposition of a $M \times N$ matrix can be executed in polynomial time $\mathcal{O}(\min(N^d, M^d))$ on a classical computer, where for the best current algorithms $2 \leq d \leq 3$. Note that for sparse or low-rank matrices, the structure can be exploited for faster routines.

7.2 The quantum linear regression algorithm

The quantum linear regression algorithm is supposed to end up in a quantum state from which the value for $\tilde{y}$ given in Equation 7.4 can be extracted by measurement. In order to harvest a potential exponential speedup the information will be processed in *amplitude encoding*. I follow the approach to assume that all classical information (i.e., the input data matrix $\mathbf{X}$, the output data vector $\mathbf{y}$ as well as the new input $\tilde{\mathbf{x}}$) is encoded into the amplitudes of three quantum states $|\psi_{\mathbf{X}}\rangle, |\psi_{\mathbf{y}}\rangle$ and $|\psi_{\tilde{\mathbf{x}}}\rangle$. Visit Section 4.2 for a dicussion of the costs of state preparation that have to be added to the runtime considerations. The number of qubits needed to represent the M training input vectors with N entries each, M training outputs and N entries of the new input is $2\lceil \log N \rceil + 2\lceil \log M \rceil$. The central advantage of the quantum algorithm arises only if it remains

polynomial in the number of qubits (or "super-efficient"), thereby taking time logarithmic in the size of the training set and the dimension of the inputs.

7.2.1 Summary

In order to 'compute' $\tilde{y}$ we need to extract and invert the singular values of $\mathbf{X}$, which are at the same time the eigenvalues of the Hermitian matrix $\mathbf{X}^\dagger\mathbf{X}$. As explained in Section 3.4.5, this can be done if we simulate a Hamiltonian corresponding to $\mathbf{X}^\dagger\mathbf{X}$, write the eigenvalues into basis encoding via phase estimation and perform a postselective amplitude update to invert the eigenvalues. For the Hamiltonian simulation to be efficient in general, $\mathbf{X}^\dagger\mathbf{X}$ has to be sparse, which poses a severe restriction on the data, and it could be argued that for sparse data, classical methods might exist to find the inverse in logarithmic time as well. This is why I will follow a different strategy and apply the density matrix exponentiation Routine 4.3.2. As a reminder, this technique shows how to efficiently simulate a Hamiltonian corresponding to an arbitrary density matrix, or to 'exponentiate a density matrix'.

The input to the algorithm are the three quantum states

$$|\psi_{\mathbf{X}}\rangle = \sum_{j=0}^{N-1}\sum_{m=0}^{M-1} x_j^{(m)}|j\rangle|m\rangle, \tag{7.5}$$

$$|\psi_{\mathbf{y}}\rangle = \sum_{\mu=0}^{M-1} y^{(\mu)}|\mu\rangle, \tag{7.6}$$

$$|\psi_{\tilde{\mathbf{x}}}\rangle = \sum_{\gamma=0}^{N-1} \tilde{x}_\gamma|\gamma\rangle, \tag{7.7}$$

where the amplitudes are real and normalised as $\sum_{m,j}(x_j^{(m)})^2 = \sum_\mu (y^{(\mu)})^2 = \sum_\gamma(\tilde{x}_\gamma)^2 = 1$. The first state represents the data matrix of training inputs and we require a number of copies of it that grows with the required accuracy of the result.

Given $|\psi_{\mathbf{X}}\rangle$ it turns out to be trivial to get a density matrix $\rho_{\mathbf{X}\mathbf{X}^\dagger}$ that is entrywise similar to $\mathbf{X}\mathbf{X}^\dagger$ and which can be 'applied' to $|\psi_{\mathbf{X}}\rangle$ in order to extract the eigenvalues of $\mathbf{X}\mathbf{X}^\dagger$. These eigenvalues can then be used to invert the singular values with known techniques, and in a last step I will use $|\psi_{\mathbf{y}}\rangle$ and $|\psi_{\tilde{\mathbf{x}}}\rangle$ to arrive at a quantum state that carries the desired output 7.4 as an off-diagonal element of a qubit. The steps and corresponding techniques are summarised below and will be explained in further detail in the remainder of this section.

Step 1:	Exponentiate $\rho_{\mathbf{X}\mathbf{X}^\dagger}$ and apply to data state using the techniques of *quantum principal component analysis* (Routine 4.3.2)
Step 2:	Extract the eigenvalues via *quantum phase estimation* (Routine 3.4.2)
Step 3:	Invert the eigenvalues through a *postselective amplitude update* (Routine 3.4.3)
Step 4:	Predict the new output with an *interference circuit* (Routine 3.4.4)

In order to make the next steps visible, one needs to reformulate $|\psi_{\mathbf{X}}\rangle$ in the eigenbasis of $\mathbf{X}\mathbf{X}^\dagger$

and $\mathbf{X}^\dagger\mathbf{X}$, which consists of the singular vectors we are looking for:

$$\begin{aligned}|\psi_{\mathbf{X}}\rangle &= \sum_{j=0}^{N-1}\sum_{m=0}^{M-1} x_j^{(m)}|j\rangle|m\rangle,\\ &= \sum_{j=0}^{N-1}\sum_{m=0}^{M-1}\sum_{r=1}^{R}\sigma_r u_{mr}v_{rj}|j\rangle|m\rangle,\\ &= \sum_{r=1}^{R}\sigma_r|\psi_{\mathbf{v}_r}\rangle|\psi_{\mathbf{u}_r}\rangle,\end{aligned}$$

where $|\psi_{\mathbf{v}_r}\rangle = \sum_{j=1}^{J}(v_j)_r|j\rangle$ and $|\psi_{\mathbf{u}_r}\rangle = \sum_{m=1}^{M}(u_m)_r|m\rangle$. This reformulation is known in quantum information as the *Schmidt decomposition.*

Now consider the corresponding density matrix $\rho_{\mathbf{X}} = |\psi_{\mathbf{X}}\rangle\langle\psi_{\mathbf{X}}|$. Excluding the $|m\rangle$ register from the description is mathematically implemented by a trace operation over that register, leading to

$$\rho_{\mathbf{X}^\dagger\mathbf{X}} = \sum_{j,j'=1}^{N}\sum_{m=1}^{M} x_j^{(m)}x_{j'}^{(m)}|j\rangle\langle j'|,$$

which is entrywise equivalent to $\mathbf{X}^\dagger\mathbf{X}$. One could alternatively trace out the j register and use $\mathbf{X}\mathbf{X}^\dagger$ with the same result, as they share the eigenvalues; the choice depends on whether $N > M$ or not.

7.2.2 Density matrix exponentiation and eigenvalue extraction (Step 1)

Density matrix exponentiation is based on the evolution of a swap operator applied to two copies of $\rho_{\mathbf{X}^\dagger\mathbf{X}}$ (i.e., two quantum systems in this state), and tracing out the second copy results in an effective simulation of a Hamiltonian that corresponds to $\rho_{\mathbf{X}^\dagger\mathbf{X}}$ up to an error of order Δt^2 (which is chosen to be small by simulating the swap operator for a short time only):

$$\mathrm{tr}_2\{e^{-iS\Delta t}(\rho_{\mathbf{X}^\dagger\mathbf{X}}\otimes\rho_{\mathbf{X}^\dagger\mathbf{X}})e^{iS\Delta t}\} = e^{-iH_\rho\Delta t} + \mathcal{O}(\Delta t^2). \tag{7.8}$$

Applying $H_{\mathbf{X}^\dagger\mathbf{X}}$ to $|\psi_{\mathbf{X}}\rangle$ evaluates to

$$\sum_{r=1}^{R}\sigma_r e^{-i\lambda_r\Delta t}|\psi_{\mathbf{v}_r}\rangle|\psi_{\mathbf{u}_r}\rangle.$$

In order to apply the quantum phase estimation technique, we need to be able to apply powers of the Hamiltonian entangled with an index register,

$$\sum_{k=1}^{K}\sum_{r=1}^{R}\sigma_r|k\rangle e^{-ik\lambda_r\Delta t}\,|\psi_{\mathbf{v}_r}\rangle|\psi_{\mathbf{u}_r}\rangle.$$

and Routine 4.3.3 shows how to use several copies of $\rho_{\mathbf{X}^\dagger\mathbf{X}}$ joined with an index register,

$$\sum_k |k\rangle\langle k|\otimes\rho_{\mathbf{X}^\dagger\mathbf{X}}\otimes\rho^{(1)}_{\mathbf{X}^\dagger\mathbf{X}}\otimes\ldots\otimes\rho^{(N)}_{\mathbf{X}^\dagger\mathbf{X}}$$

and simulate a product of 2-qubit swap operators which swap the first state $\rho_{\mathbf{X}^\dagger\mathbf{X}}$ with the gth copy $\rho^{(g)}_{\mathbf{X}^\dagger\mathbf{X}}$, where the product runs up to copy n for each term in the superposition:

$$\frac{1}{2^q}\sum_{k=1}^{K}|k\rangle\langle k|\otimes\prod_{g=1}^{k}e^{-iS_g\Delta t}.$$

The density matrix exponentiation routine requires a large amount of copies of the input state $|\psi_{\mathbf{X}}\rangle$ for the amended version that prepares the state for phase estimation in the next step. More precisely, if ϵ is the error we allow for the final state (i.e. $||\rho_{\text{desired}}-\rho_{\text{final}}||<\epsilon$), the number of copies is of the order $\mathcal{O}(\epsilon^{-3})$. Only if $\mathbf{X}^\dagger\mathbf{X}$ is dominated by a few large eigenvalues, the routine is logarithmic in the dimensions of the dataset (i.e. the number of amplitudes): It takes time $t=\mathcal{O}(1/\delta)$ to simulate e^{iHt} for a Hamiltonian H up to error δ [90], and with the trick from [140] it takes time t^2 to do the same for $e^{i\rho t}$. This means that if we want to resolve relatively uniform eigenvalues of the order of $1/N$, time grows quadratically with N and the logarithmic dependency is lost. This means that the method is only "super-efficient" if the density matrix has a low-rank approximation, which is true if there is a large amount of redundancy in the dataset. One important point to consider here is that for low-rank-approximations of the data matrix, also classical algorithms can be optimised to perform a lot better at matrix inversion. A thorough comparison of the nature of the speedup would therefore be an important future investigation.

7.2.3 Eigenvalue extraction (Step 2)

The quantum phase estimation algorithm applied to the result of the previous step leads to a state

$$\sum_{r=1}^{R}\sigma_r|\psi_{\mathbf{v}_r}\rangle|\psi_{\mathbf{u}_r}\rangle|\lambda_r\rangle,$$

in which the eigenvalues $\lambda_r=(\sigma_r)^2$ of $\rho_{\mathbf{x}^\dagger\mathbf{x}}$ are approximately encoded in the τ qubits of an extra third register that was initially in the ground state.

7.2.4 Eigenvalue inversion (Step 3)

A postselective amplitude update then yields

$$\sum_{r=1}^{R}\sigma_r|\psi_{\mathbf{v}_r}\rangle|\psi_{\mathbf{u}_r}\rangle|\lambda_r\rangle\left(\sqrt{1-\left(\frac{c}{\lambda_r}\right)^2}|0\rangle+\frac{c}{\lambda_r}|1\rangle\right).$$

The constant c is chosen so that the inverse eigenvalues are not larger than 1, which is given if it is smaller than the smallest nonzero eigenvalue $\lambda^{\min}$ of $\mathbf{X}^\dagger\mathbf{X}$, or equivalently, the smallest nonzero squared singular value $(\sigma^{\min})^2$ of $\mathbf{H}$. We perform a conditional measurement on the ancilla qubit, only continuing the algorithm ('accepting') if the ancilla is in state $|1\rangle$ (else the entire procedure has to be repeated). Considering that $(\sigma_r)^2=\lambda_r$ and denoting the probability of acceptance by

$p(1) = \sum_r \left|\frac{c}{\lambda_r}\right|^2$ the result of this step is

$$\frac{1}{\sqrt{p(1)}} \sum_{r=1}^{R} \frac{c}{\sigma_r} |\psi_{\mathbf{v}_r}\rangle |\psi_{\mathbf{u}_r}\rangle |\lambda_r\rangle.$$

Uncomputing and discarding the eigenvalue register gives

$$\frac{1}{\sqrt{p(1)}} \sum_{r=1}^{R} \frac{c}{\sigma_r} |\psi_{\mathbf{v}_r}\rangle |\psi_{\mathbf{u}_r}\rangle,$$

which already contains all necessary elements for the final solution.

The singular value inversion procedure determines the runtime's dependency on the condition number of $\mathbf{X}$, $\kappa = \sigma^{\max}(\sigma^{\min})^{-1}$. The probability to measure the ancilla in the excited state is given by

$$p(1) = \sum_r \left|\frac{c}{\lambda_r}\right|^2 \leq R \left|\frac{\lambda^{\min}}{\lambda^{\max}}\right|^2 = R\kappa^{-4},$$

which means one needs on average less than κ^4 tries to accept the conditional measurement. Amplitude amplification as in [90, 126] reduces this to a factor of $\mathcal{O}(\kappa^2)$ in the runtime, which can become significant for matrices that are close to being singular.

7.2.5 Prediction (Step 4)

In order to calculate the new output $\tilde{y}$, the inner products between $|\psi_{\mathbf{v}_r}\rangle$ and $|\psi_{\tilde{\mathbf{x}}}\rangle$ as well as between $|\psi_{\mathbf{u}_r}\rangle$ and $|\psi_{\mathbf{y}}\rangle$ have to be evaluated. As outlined in Section 3.4.6, the usual strategy would be a swap routine which reveals the abolute square of an inner product. The problem with the swap routine is that it does not reveal the sign of the inner product, which might be important for the task at hand, especially if the data is preprocessed to a zero mean. A simple trick can help, which is to 'add an amplitude' fixed to the value $\sqrt{0.5}$ to all quantum states, which will in most applications not even require to add a qubit when the dimension of the Hilbert space 2^n is larger than the dimension of the data matrix and we 'fill up' with zero amplitudes. An inner product of such two quantum states will always have a term of 0.5 stemming from the product of the two additional amplitudes, and the sum of all other terms has to be between $[-0.5, 0.5]$. Therefore, the additional amplitude shifts the result of the inner product from the interval $[-1, 1]$ to the interval $[0, 1]$ and reveals negative outcomes.

A slightly different strategy can be followed if we can implement the entire routine above including state preparation controlled by an additional qubit to get

$$\frac{1}{\sqrt{p(1)}} \sum_{r=1}^{R} \frac{c}{\sigma_r} |\psi_{\mathbf{v}_r}\rangle |\psi_{\mathbf{u}_r}\rangle |0\rangle + |\psi_{\mathbf{y}}\rangle |\psi_{\tilde{\mathbf{x}}}\rangle |1\rangle.$$

If we trace out all registers except from the ancilla, the offdiagonal elements ρ_{12}, ρ_{21} of the ancilla's density matrix read

$$\frac{c}{2\sqrt{p(1)}} \sum_r \frac{1}{\sigma_r} \sum_j (v_j)_r \tilde{x}_j \sum_m (u^m)_r y^{(m)},$$

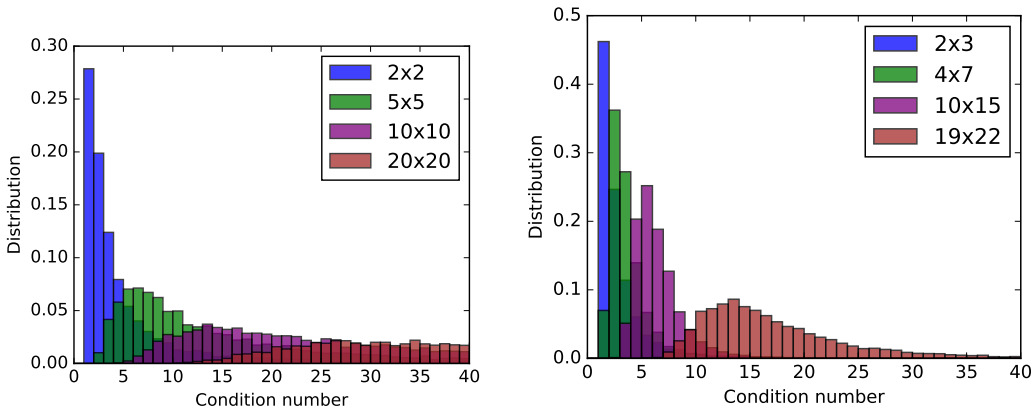


Figure 7.1: Distribution of condition numbers for matrices of different dimensions with entries drawn uniformly at random from $[-1, 1]$. For square matrices (left) the mean of the distribution increases with the dimension, while for rectangular matrices (right) the increase is much slower, and curiously enough in all simulations performed sublinear in the smallest dimension. The growth of the condition number with the dataset has to be taken into account if the "super-efficient" speedup is to be maintained. Note that the distributions remain the same if sparse or symmetric matrices are generated.

and contain the desired result (7.4) up to a known normalisation factor $\frac{c}{2\sqrt{p(1)}}$ as well as the rescaling factor $|\mathbf{X}|^{-1}|\tilde{\mathbf{x}}|^{-1}|\mathbf{y}|^{-1}$ stemming from the initial normalisation of the data that allowed us to encode it into quantum states. Whichever way is chosen, the prediction step is linear in the number of qubits used.

In summary, the upper bound for the runtime of the quantum linear regression algorithm can be roughly estimated as $\mathcal{O}(\log N\kappa^2\epsilon^{-3})$ with the error ϵ, condition number κ and input dimension N. For the swap test, a factor $\log M$ for the swap operator has to be considered. Remember that this does not include the costs of quantum state preparation in case the algorithm processes classical information. The quantum algorithm is "super-efficient" if the condition number scales logartihmically in the dimensions of the dataset, which as Figure 7.1 suggests is not always the case. Compared to the previous result by Wiebe, Braun and Lloyd, this version boasts an improvement of a factor κ^{-4} whereas the dependence on the accuracy is worse by a factor ϵ^{-2}. The algorithm can efficiently be applied to nonsparse, but low rank approximations of the matrix $\mathbf{X}^\dagger\mathbf{X}$.

7.3 Extending the model

7.3.1 How to add regularisation

When applying linear regression in practise, a regularisation term is usually added to the objective function in order to penalise large parameters (similar to weight decay or pruning in neural networks),

$$\hat{\mathbf{w}} = \min_{\mathbf{w}} |\mathbf{X}\mathbf{w} - \mathbf{y}|^2 + \alpha||\mathbf{w}||.$$

The norm $||\mathbf{w}||$ can refer to the L_1 norm $\sum_i w_i^2$ (*ridge-regression*) or L_2 norm $\sum_i |w_i|$ (*LASSO* procedure), in general leading to different optimal solutions.[2] I will focus on ridge regression here.

The extra term changes the pseudoinverse to

$$\mathbf{X}^+ = (\mathbf{X}^\dagger \mathbf{X} + \alpha \mathbb{I})^{-1} \mathbf{X}^\dagger,$$

and inserting again the singular value decomposition of $\mathbf{X}$ leads to the slightly different estimator of the parameter vector,

$$\hat{\mathbf{w}} = \sum_{r=1}^{R} \frac{\sigma_r}{\sigma_r^2 - \alpha} \mathbf{u}_r^T \mathbf{y} \mathbf{v}_r, \tag{7.9}$$

It is not difficult to amend the original quantum linear regression routine to cater for the additional term in the scalar. In Step 3, when rotating the ancilla conditioned on the eigenvalue register, rotate the ancilla according to the eigenvalue plus regularisation factor instead:

$$\sum_{r=1}^{R} \sigma_r |v_r\rangle |u_r\rangle |\lambda_r\rangle \left(\sqrt{1 - \left(\frac{c}{\lambda_r + \alpha} \right)^2} |0\rangle + \frac{c}{\lambda_r + \alpha} |1\rangle \right).$$

Since α is positive, this increases the probability of accepting the conditional measurement, $p_{\text{reg}}(1) = \sum_r \left| \frac{c}{\lambda_r + \alpha} \right|^2$, which is a similar effect to increasing the numerical stability in the classical computations with regularisation. Both lead to the algorithm being less dependent on the condition number.

7.3.2 How to implement feature maps

As the name says, linear regression is based on a linear model with a likewise linear decision boundary and derives its true power from the kernel trick. As a reminder, the kernel trick refers to the procedure of mapping the input vectors to a higher dimensional space through a nonlinear feature map and performing the classification in this space, where the data becomes linear separable. This is how a linear model can fit nonlinear data.

In the routine outlined above, applying a feature map means to replace the kernel matrix $\rho_{\mathbf{x}\mathbf{x}^\dagger}$ with other Gram matrices.

7.3.2.1 Linear kernels

A linear or simple dot kernel is a function $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$. The corresponding kernel (Gram) matrix for input data vectors $x^{(m)}, m = 1...M$ has entries $K_{mm'} = (\mathbf{x}^{(m)})^T \mathbf{x}^{(m')}$. The procedure to prepare a density matrix $\rho_{\mathbf{K}}$ that entrywise represents the kernel matrix from a quantum state encoding the training inputs in its amplitudes has been the one used above. To recapitulate, assume the (normalised) training data is given in amplitude encoding,

$$|D\rangle = \sum_{m=1}^{M} \sum_{k=1}^{N} x_k^{(m)} |k\rangle |m\rangle,$$

[2] LASSO prefers sparse parameter vectors.

the corresponding density matrix is

$$|D\rangle\langle D| = \sum_{m,m'=1}^{M} \sum_{k,k'=1}^{N} x_k^{(m)} (x_{k'}^{(m')})^* |k\rangle\langle k'||m\rangle\langle m'|.$$

Taking the trace over the $|k\rangle$ register yields the desired density matrix $\rho_{\mathbf{K}} = \mathrm{tr}_k |D\rangle\langle D|$,

$$\rho_{\mathbf{K}} = \sum_{m,m'=1}^{M} \sum_{k=1}^{N} x_k^{(m)} (x_k^{(m')})^* |m\rangle\langle m'|.$$

7.3.2.2 Homogeneous polynomial kernels

A polynomial kernel of order d is a function $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^d$. The corresponding kernel (Gram) matrix has entries $K_{mm'} = ((\mathbf{x}^{(m)})^T \mathbf{x}^{(m')})^d$ and can be constructed by using d copies of $|\psi_{\mathbf{x}^{(m)}}\rangle = \sum_{k=1}^{N} x_k^{(m)} |k\rangle$ [126]. Given

$$|D\rangle = \sum_{m=1}^{M} (|\psi_{\mathbf{x}^{(m)}}\rangle \otimes |\psi_{\mathbf{x}^{(m)}}\rangle \otimes \ldots \otimes |\psi_{\mathbf{x}^{(m)}}\rangle) \otimes |m\rangle,$$

(where the tensor products are made explicit for now), the corresponding density matrix becomes

$$|D\rangle\langle D| = \sum_{m,m'=1}^{M} \prod_{i=1}^{d} \left[\sum_{k_i,k_i'=1}^{N} x_{k_i}^{(m)} (x_{k_i'}^{(m')})^* \right] |k_1\rangle\langle k_1'| \otimes \ldots \otimes |k_d\rangle\langle k_d'| \otimes |m\rangle\langle m'|.$$

The trace over all $|k_i\rangle$ registers now yields

$$\rho_{\mathbf{K}} = \sum_{m,m'=1}^{M} (\sum_{k=1}^{N} x_k^{(m)} (x_k^{(m')})^*)^d |m\rangle\langle m'|.$$

Replacing the density matrix $\rho_{\mathbf{X}^T\mathbf{X}}$ above with this kernel matrix implicitly implements a feature map into a space of polynomials of degree d.

7.3.2.3 Translation invariant kernels

Although the dot-product kernel functions above are in use, a lot of theoretical analysis considers translation invariant kernels where $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(|\mathbf{x} - \mathbf{x}'|)$. Important examples are exponential versions such as the Gaussian or radial basis function kernel $\exp(1/2|\mathbf{x} - \mathbf{x}'|^2)$ and rational functions such as the triangular kernel $1 - |\mathbf{x} - \mathbf{x}'|$ and the Epanechnikov kernel $3/4(1 - |\mathbf{x} - \mathbf{x}'|^2)$, both being zero outside of the interval $[-1, 1]$. Other covariance functions are discussed in Rasmussen & Williams' textbook on Gaussian Processes [30].

It is an interesting question whether we can also implement translation invariant kernels with density matrices. This question has very recently been addressed by Chatterjee and Yu [216], who encode features into canonical coherent states whose inner product naturally implements in a radial kernel function. It has also been shown here that with the swap test Routine 3.4.4 that for units vectors, there is a close relationship between the inner product and the difference between

two unit vectors $\mathbf{x}$ and $\mathbf{x}'$:

$$\begin{aligned} |\mathbf{x} - \mathbf{x}'|^2 &= \sum_i (x_i - x_i')^2 \\ &= 2 - 2\sum_i x_i x_i' \\ \Leftrightarrow -(1 - 0.5|\mathbf{x}' - \mathbf{y}|^2) &= (\mathbf{x}')^T \mathbf{y} \end{aligned}$$

This implies that the dot product kernel is in this case similar to the negative Epanechikov kernel, which looks very much like a semi-circle. Which other kernels are natural for the density matrix of a quantum system? Can we combine the theory of so called reproducing kernel Hilbert spaces with quantum mechanics to our avail? This is a very promising question for further research.

Chapter 8

Quantum gradient descent algorithm

This chapter is based on work done in collaboration with Dr Patrick Rebentrost and Prof Seth Lloyd from the Massachusetts Institute of Technology that is available as a pre-print (Reference [217]). The idea was developed by Patrick Rebentrost, Seth Lloyd and myself, while Patrick Rebentrost and I were with equal shares responsible to work out and formulate the quantum algorithm in detail. Patrick Rebentrost was particularly involved in the error analysis in the Appendix, while I wrote the computer simulations for the analysis and plots. The presentation in this chapter is written by myself.

The previous chapter on least squares regression implicitly dealt with an unconstrained *quadratic programme* that can be expressed as an objective function $\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} + 2\mathbf{A}\mathbf{x}\mathbf{y} + \mathbf{y}^2$ and a closed form equation for the solution was given. Quadratic programmes of the form $\mathbf{x}\mathbf{B}\mathbf{x}$ can be very difficult to solve (i.e., proven to be NP-hard [218]) if $\mathbf{B}$ is not positive definite (which it always is for $\mathbf{B} = \mathbf{A}^T\mathbf{A}$), and the function therefore non-convex. Other machine learning methods can reveal even more complicated optimisation landscapes, for example the notorious deep neural networks. In such cases iterative methods that perform a stepwise search for the optimum can be the only possible solution method. The most common are gradient descent methods, using local information on gradients in each step. Unfortunately there is only little theory about overall convergence times and numerical stability and benchmark studies are usually done by comparing the performance of a machine learning algorithm on several standard datasets in practise.

The quantum machine learning community has so far widely avoided the topic of iterative optimisation methods such as gradient descent. For example, proposals of quantum neural networks are mostly limited to proposing a feed-forward mechanism and leaving the central question of training open, or suggests classical parameter fitting for the quantum setup. An important reason for the reluctance is the difficulty to come up with a clever quantum routine that extracts information on the best direction for a local search and updates a quantum state accordingly, and which exploits the laws of quantum mechanics to achieve a speed-up of some sorts. The attempt is thereby not new; the research field of quantum optimisation has produced a number of 'quantisations' of classical algorithms, some of which have been mentioned before.

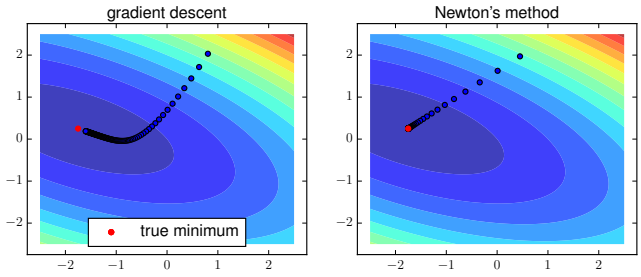


Figure 8.1: Gradient descent and Newton's method for a quadratic objective function $0.5\boldsymbol{\theta}^T A\boldsymbol{\theta} + \mathbf{b}^T\boldsymbol{\theta}$ with $A = [[0.2, 0.2], [0.2, 0.6]]$ and $\mathbf{b} = (0.3\ 0.2)^T$. The stepsize is $\eta = 0.2$ and 100 steps are performed. It is obvious that while gradient descent follows the gradients, Newton's method finds a more direct route to the minimum.

This chapter is (to my knowledge) a first approach to find a quantum version of iterative optimisation algorithms such as gradient descent and Newton's method. The objective function considered here is a homogeneous polynomial under unit sphere constraints, but the principle may be applicable for other problems as well. It is based on the amplitude encoding strategy, and a vector encoded in a quantum state is updated in every step according to a 'normalised' variation of the classical method (i.e., it reproduces the classical result but harvests the potential advantages of exponentially compact information encoding). A central caveat is that the algorithm 'uses' a large number of copies of a quantum state in order to find the next point, which is infeasible for the long iteration cycles of common gradient descent methods. Newton's method has more favourable convergence properties known to converge quadratically close to a minimum. The scheme could therefore be interesting for local optimisation, such as fine tuning after pre-training.

8.1 Classical problem formulation

Before introducing the rather technical quantum algorithm, I want to briefly discuss gradient descent and Newton's method for optimisation, introduce the objective function that serves as an example here, and formulate important derivatives as well as normalisation constraints.

8.1.1 Standard gradient descent and Newton's method

The idea of gradient descent methods is to start with a 'good guess' for a candidate that solves the optimisation problem, and then to iteratively replace the candidate with a better one (also known as fixed point iteration). In each update, information on the gradient of the objective function at the current candidate position is used to find the direction in which the objective function decreases (for minimisation).

Let $o : \mathbb{R}^{\mathbb{N}} \to \mathbb{R}$ be the objective function one seeks to minimise, and $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^N$ the initial candidate for the minimum. An update is performed according to

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla o(\boldsymbol{\theta}^{(t)}), \tag{8.1}$$

where $\eta > 0$ is a hyper-parameter called the learning rate that may change with the number of steps t. A (local) minimum is found if the gradient vanishes and $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$.

Newton's method adapts this procedure by multiplying the inverse Hessian matrix to the gradient,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta\, \mathbf{H}^{-1} \nabla o(\boldsymbol{\theta}^{(t)}). \tag{8.2}$$

The Hessian matrix has entries $H_{ij} = \frac{\partial^2 o(\boldsymbol{\theta}^{(t)})}{\partial\theta_i \partial\theta_j}$ where $o(\boldsymbol{\theta}^{(t)})$ is evaluated at the current candidate point $\boldsymbol{\theta}^{(t)}$. By taking the second derivative into account Newton's method evaluates curvature information on top of the gradients and therefore tends to follow a more direct path to the minimum (see Figure 8.1).

8.1.2 The objective function

I will only consider homogeneous polynomial optimisation functions here, which are generalisations of linear and quadratic programmes. To translate smoothly to the quantum algorithm, the chosen notation is based on tensor products and therefore a little unusual.

Consider an unconstrained linear programme minimising the vector-valued objective function

$$o_1(\boldsymbol{\theta}) = \mathbf{q}^T \boldsymbol{\theta} + \theta_0,$$

with $\boldsymbol{\theta}, \mathbf{q} \in \mathbb{R}^N$, $r \in \mathbb{R}$. One can increase the order of the polynomial to an unconstrained *quadratic progamme*,

$$o_2(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta} + \mathbf{q}^T \boldsymbol{\theta} + \theta_0,$$

The next higher order would require $\mathbf{A}$ to be a 3-tensor,

$$o_3(\boldsymbol{\theta}) = \frac{1}{3} \sum_{i,j,k=1}^{N} \theta_i \theta_j \theta_k b_{ijk} + \frac{1}{2} \sum_{i,j,k=1}^{N} \theta_i \theta_j a_{ij} + \sum_{i=1}^{N} \theta_i q_i + \theta_0.$$

If we only consider the highest order term and require that $k = 2p$ is an even number we get the family of even homogenous polynomials, which can be written as

$$o_k(\boldsymbol{\theta}) = \frac{1}{2p} \sum_{i_1,\ldots,i_{2p}=1}^{N} a_{i_1 \ldots i_{2p}} \theta_{i_1} \ldots \theta_{i_{2p}}.$$

This is the objective function considered here. In machine learning one often turns non-homogeneous polynomials into this form by demanding that the first parameter is clamped to one (which has been mentioned for linear model when the parameter w_0 was included in the $\mathbf{w}^T\mathbf{x}$ term). This is unfortunately not possible when iteratively renormalising the vector as done in the projected and later in the quantum methods. One can however think of tricks to introduce minor 'non-homogeneities'.

Now consider a slightly different formulation. The even orders $k = 2p$ can be written in the form of a quadratic programme if we go into a space that is created by taking p tensor products of the

$\mathbb{R}^N$,

$$o_{2p}(\boldsymbol{\theta}) = \frac{1}{2} \underbrace{\boldsymbol{\theta}^T \otimes \ldots \otimes \boldsymbol{\theta}^T}_{p \text{ times}} \mathbf{A} \underbrace{\boldsymbol{\theta} \otimes \ldots \otimes \boldsymbol{\theta}}_{p \text{ times}}, \tag{8.3}$$

with $\boldsymbol{\theta} \in \mathbb{R}^N$ and $\mathbf{A}$ containing the coefficients. The tensor $\mathbf{A} \in \mathbb{R}^N \otimes \ldots \otimes \mathbb{R}^N$ can always be written this way, i.e. as a sum of tensor products of $N \times N$ matrices $\mathbf{A}_i^{(\alpha)}$, $i = 1...p$,

$$\mathbf{A} = \sum_{\alpha} \mathbf{A}_1^{(\alpha)} \otimes \ldots \otimes \mathbf{A}_p^{(\alpha)}. \tag{8.4}$$

To illustrate this notation and its scope, consider the following example:

Example 8.1.1: Polynomial objective function for $p = 2, N = 2$

For a 2 dimensional input domain, a 4th order polynomial can be represented in the form of Equation (8.3) as

$$\begin{aligned}
o_4(\boldsymbol{\theta}) &= \frac{1}{2} (\theta_1, \theta_2) \otimes (\theta_1, \theta_2) \begin{pmatrix} a_{1111} & a_{1112} & a_{1211} & a_{1212} \\ a_{1121} & a_{1122} & a_{1221} & a_{1222} \\ a_{2111} & a_{2112} & a_{2211} & a_{2212} \\ a_{2121} & a_{2122} & a_{2221} & a_{2222} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \otimes \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \\
&= \frac{1}{2} (\theta_1\theta_1, \theta_1\theta_2, \theta_2\theta_1, \theta_2\theta_2) \begin{pmatrix} a_{1111} & a_{1112} & a_{1211} & a_{1212} \\ a_{1121} & a_{1122} & a_{1221} & a_{1222} \\ a_{2111} & a_{2112} & a_{2211} & a_{2212} \\ a_{2121} & a_{2122} & a_{2221} & a_{2222} \end{pmatrix} \begin{pmatrix} \theta_1\theta_1 \\ \theta_1\theta_2 \\ \theta_2\theta_1 \\ \theta_2\theta_2 \end{pmatrix} \\
&= \frac{1}{2} a_{1111} \theta_1^4 \\
&\quad + \frac{1}{2}(a_{1112} + a_{1121} + a_{1211} + a_{2111}) \theta_1^3\theta_2 \\
&\quad + \frac{1}{2}(a_{1122} + a_{1212} + a_{1221} + a_{2112} + a_{2121} + a_{2211}) \theta_1^2\theta_2^2 \\
&\quad + \frac{1}{2}(a_{1222} + a_{2122} + a_{2212} + a_{2221}) \theta_1\theta_2^3 \\
&\quad + \frac{1}{2} a_{2222} \theta_2^4
\end{aligned}$$

If we could clamp $\theta_2 = 1$ we would get a polynomial over the $N - 1 = 1$-dimensional input space with all orders up to fourth, namely $1, \theta_1, \theta_1^2, \theta_1^3, \theta_1^4$.

Equation (8.3) constitutes the type of objective function for which the quantum gradient descent algorithm will be formulated.

8.1.3 Normalisation constraints

When encoding the classical vectors $\boldsymbol{\theta}$ as quantum states in the next section, an additional constraint appears:

$$\sum_{i=1}^{N} \theta_i^2 = 1,$$

or $\boldsymbol{\theta}^T\boldsymbol{\theta} = 1$. In the optimisation literature this is known as a *spherical constraint* or *orthogonality constraint*. Applications of such optimisation problems appear in image and signal processing,

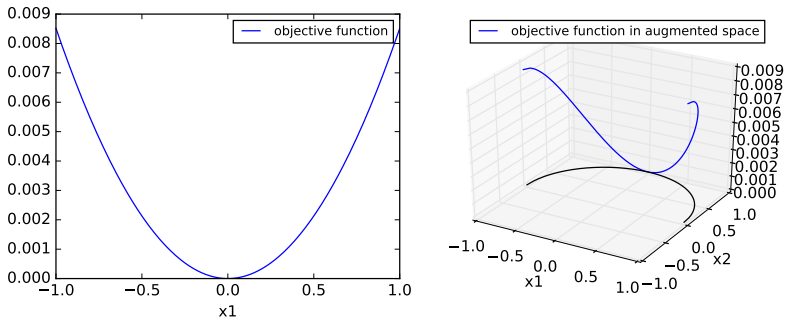


Figure 8.2: Embedding of a quadratic objective function over the interval $[-1, 1]$ onto the unit circle of a 2-dimensional space. It is an interesting question whether such embeddings can transform a L_2 regularisation into a 'normalisation condition'.

biomedical engineering, speech recognition and even quantum mechanics [219].[1] For the purpose of machine learning, spherical optimisation is not very common, while L_2-ball optimisation with the constraint $\boldsymbol{\theta}^T\boldsymbol{\theta} \leq 1$ appears frequently as a regularisation method. The constraint ensures that the length of the parameter vector does not exceed a certain size. It needs to be established in future research what effects replacing $\boldsymbol{\theta}^T\boldsymbol{\theta} \leq 1$ with $\boldsymbol{\theta}^T\boldsymbol{\theta} = 1$ will have on regularisation. Intuitively, this reduces the risk that the parameters vanish and might be interesting for recurrent neural networks, where length-preserving training methods have recently been proposed [222]. Furthermore, in cases where the local minimum of the overall objective function lies outside of the unit ball, L_2 regularisation would find the constrained minimum on the unit sphere in any case, which coincides with the case of unit sphere constraints. For cases in which unit sphere constraints have unfavourable properties it would be interesting to investigate whether the spherical constraints can be mapped to ball constraints in a lower dimension (as depicted in Figure 8.2). These open questions are left for future investigations.

In general, constrained optimisation problems can be solved with *gradient projection methods* [223, 224], in which after each gradient descent update the result is projected to the feasible set, here given by $S = \{\boldsymbol{\theta} \in \mathbb{R}^N | \boldsymbol{\theta}^T\boldsymbol{\theta} = 1\}$. A proof of convergent in convex basins around a minimum is given in the Appendix B.3. The quantum method below will recover the exact behaviour of a projected method, but the normalisation step will be generically performed by measurements on the quantum system.

The objective function defined above together with the normalisation constraint shapes a very specific type of optimisation problem, and it is important to have a look at typical problem instances here. Homogeneous polynomials have the property of a vanishing gradient in the origin, which means that the origin is either a minimum, maximum or saddle point. For quadratic forms $o(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{A}\boldsymbol{\theta}$ the projected Newton's method is not expected to perform well, since $H^{-1}\nabla o(\boldsymbol{\theta}) = \boldsymbol{\theta}$, and the search direction is consequently parallel to the current vector and no steps are taken, as seen in Figure 8.3. In comparison, the projected gradient descent method performs as desired. This problem can be rectified by including a bias term, $o(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{A}\boldsymbol{\theta} + \mathbf{b}^T\boldsymbol{\theta}$, using a further vector

[1] The solution of optimising homogeneous polynomials under spherical constraints has been shown to be NP-hard for $d = 3$, and very difficult to approximate [220]. Beyond this case, only little is known about the complexity of spherical optimisation [221].

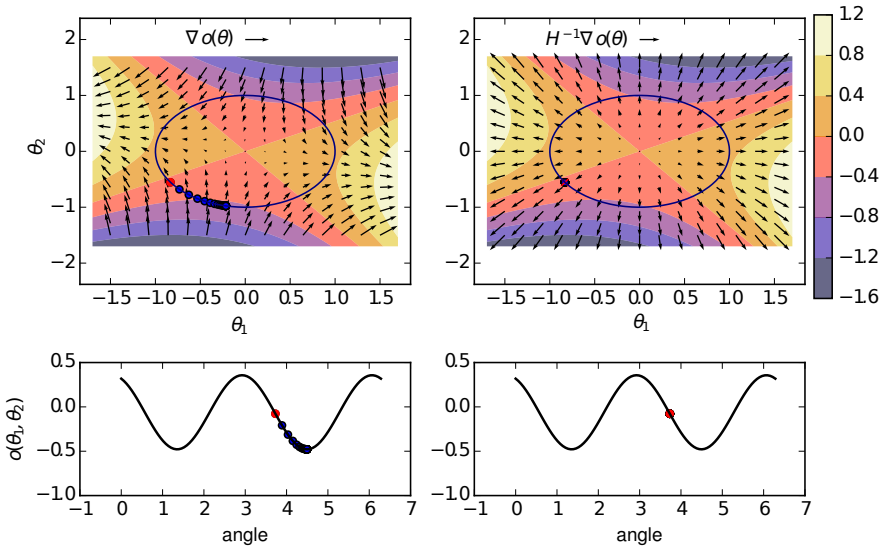


Figure 8.3: Projected gradient descent and projected Newton's method for quadratic optimisation under unit sphere constraints (i.e., the solution is constrained to the circle). Below is the function on the feasible set only, with the angle staring from position $(1,0)^T$. The parameters are chosen as $K = 1, p = 1, N = 2$ and the objective function is $o(\boldsymbol{\theta}) = \boldsymbol{\theta}\mathbf{A}\boldsymbol{\theta}$ with $A = [[0.6363, -0.7031], [0.0, -0.8796]]$ and the initial point (red dot) is chosen as $\theta_{\text{init}} \approx [-0.83, -0.55]$. The number of steps is $T = 20$. One can see that Newton's method struggles to find the minimum on the feasible set plotted below, since it would naturally update the current vector towards the origin, which is not possible under the normalisation constraints.

addition in the quantum algorithm below. With this trick, we can reproduce the example in Figure 8.1 as shown in Figure 8.4. The Figure also reveals that the objective function on the unit sphere still exhibits an interesting landscape for optimisation, and the problem is therefore non-trivial. However, it would be interesting to open the method to other objective functions, and this chapter shall serve as a starting point for future investigations.

8.1.4 Gradient and Hessian of the objective function

For the gradient descent method we need to determine the first derivative of the objective function. Using the decomposition of $\mathbf{A}$ as in Eq. 8.4, the derivative reads

$$\nabla o(\boldsymbol{\theta}) = \frac{1}{2}\sum_{\alpha=1}^{K}\sum_{j=1}^{p}\left(\prod_{i\neq j}\boldsymbol{\theta}^\dagger \mathbf{A}_i^{(\alpha)}\boldsymbol{\theta}\right)\left((\mathbf{A}_j^{(\alpha)})^T + \mathbf{A}_j^{(\alpha)}\right)\boldsymbol{\theta}. \tag{8.5}$$

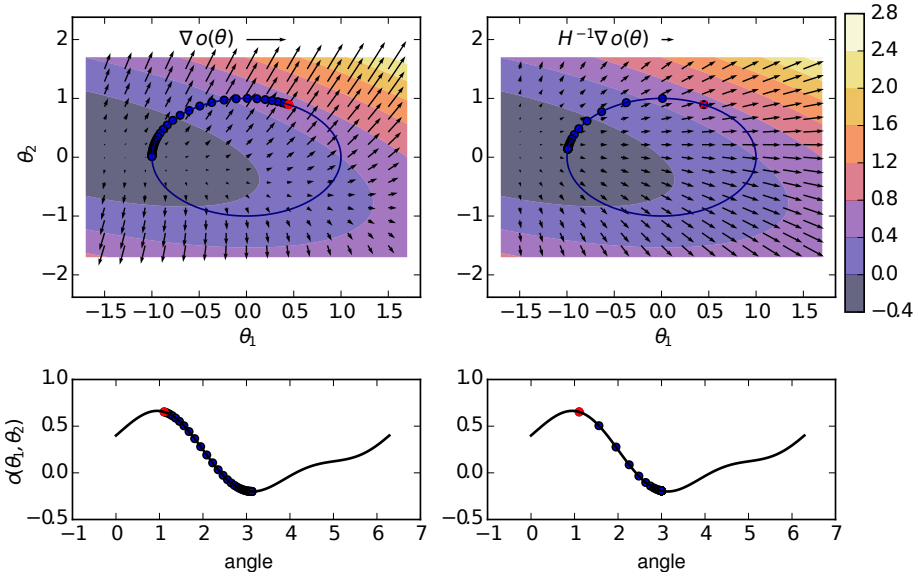


Figure 8.4: Projected descent methods for inhomogeneous polynomial. The parameters are chosen as in Figure 8.1 and shows how an inhomogeneity can make Newton's function perform well for quadratic objective functions.

Proof. Express $\mathbf{A}$ according to Equation 8.4:

$$\begin{aligned}\frac{\partial}{\partial\theta_b}\frac{1}{2}\,\boldsymbol{\theta}^\dagger\otimes\ldots\otimes\boldsymbol{\theta}^\dagger\mathbf{A}\,\boldsymbol{\theta}\otimes\ldots\otimes\boldsymbol{\theta} &= \frac{\partial}{\partial\theta_b}\sum_\alpha\prod_{i=1}^{p}\boldsymbol{\theta}^\dagger\mathbf{A}_i^\alpha\,\boldsymbol{\theta}\\ &= \frac{1}{2}\sum_\alpha\sum_j\left(\prod_{i\neq j}\sum_{st}(a_{st})_i^\alpha\theta_s\theta_t\right)\sum_{mn}(a_{mn})_j^\alpha\frac{\partial}{\partial\theta_b}(\theta_m\theta_n)\\ &= \sum_\alpha\sum_j\Big(\prod_{i\neq j}\underbrace{\sum_{st}(a_{st})_i^\alpha\theta_s\theta_t}_{\boldsymbol{\theta}^\dagger\mathbf{A}_i^{(\alpha)}\boldsymbol{\theta}}\Big)\sum_m\frac{1}{2}\left((a_{mb})_j^\alpha+(a_{bm})_j^\alpha\right)\theta_m\end{aligned}$$

□

Hence, to calculate the derivative at a given point $\boldsymbol{\theta}$ one has to apply a matrix

$$\mathbf{D}=\frac{1}{2}\sum_\alpha\sum_j\left(\prod_{i\neq j}\boldsymbol{\theta}^\dagger\mathbf{A}_i^{(\alpha)}\boldsymbol{\theta}\right)\left((\mathbf{A}_j^{(\alpha)})^T+\mathbf{A}_j^{(\alpha)}\right) \tag{8.6}$$

to the vector $\boldsymbol{\theta}$. This is crucial for the quantum algorithm, in which we will apply a corresponding quantum operator D to a quantum state vector representing the current point.

The Hessian matrix can be expressed as

$$\mathbf{H} = \frac{1}{2}\sum_{\alpha=1}^{K}\sum_{j=1}^{p}\left[\sum_{\substack{i=1\\ i\neq j}}^{p}\left(\prod_{\substack{k=1\\ k\neq i}}^{p}\boldsymbol{\theta}^{\dagger}\mathbf{A}_k^{(\alpha)}\boldsymbol{\theta}\right)\left((\mathbf{A}_i^{(\alpha)})^T+\mathbf{A}_i^{(\alpha)}\right)\boldsymbol{\theta}\,\boldsymbol{\theta}^T\left((\mathbf{A}_j^{(\alpha)})^T+\mathbf{A}_j^{(\alpha)}\right)+\left(\prod_{\substack{i=1\\ i\neq j}}^{p}\boldsymbol{\theta}^{\dagger}\mathbf{A}_i^{(\alpha)}\boldsymbol{\theta}\right)\left((\mathbf{A}_j^{(\alpha)})^T+\mathbf{A}_j^{(\alpha)}\right)\right], \quad (8.7)$$

and will likewise be translated into a quantum operator H.

Proof. Similar to above.

□

8.2 Quantum gradient descent algorithm

Although this Section will deal with many technicalities, the basic idea of the quantum gradient descent routine is simple. Two procedures are required, one that shows how an update in the iterative optimisation method is performed by turning a quantum state slightly away from the previous state $|\psi_{\boldsymbol{\theta}^{(t)}}\rangle$ to a state $|\psi_{\boldsymbol{\theta}^{(t)}}\rangle - \eta|\psi_{\nabla o}\rangle$ with a small parameter η that can be interpreted as the learning rate. This will be shown in the next subsection. The second procedure shows how to implement an operator D that maps a quantum state representing the current point to the gradient of the objective function at that point, $D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle \to |\psi_{\nabla o}\rangle$. This will be based on a novel extension of the density matrix exponentiation technique of Routine 4.3.2.

8.2.1 Outline of one step in the descent method

Let $\boldsymbol{\theta}^{(t)} = (\theta_1^{(t)}, ..., \theta_N^{(t)})^T$ be the choice of our candidate for $\boldsymbol{\theta}$ after the tth update or iteration, and $\sum_i |\theta_i^{(t)}|^2 = 1$. I will abbreviate the gradient at $\nabla o(\boldsymbol{\theta}^{(t)})$ time step t by $\nabla^{(t)}$. Prepare a quantum system in the state

$$(\cos\gamma\,|0\rangle - \mathrm{i}\,\sin\gamma\,|1\rangle)\,|\psi_{\boldsymbol{\theta}^{(t)}}\rangle, \quad (8.8)$$

where γ is a small external parameter, the first register contains a single ancilla qubit, and the second register contains the state

$$|\psi_{\boldsymbol{\theta}^{(t)}}\rangle = \sum_{i=0}^{N}\boldsymbol{\theta}_i^{(t)}|i\rangle.$$

Let D be the operator[2] that transforms the current point into the gradient at the current point, $D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle = |\psi_{\nabla^{(t)}}\rangle$. Applying D (as specified in Subroutine I below) conditioned on the ancilla being in state 1 results in

$$|\psi\rangle = \frac{1}{\sqrt{p_D}}\left(\cos\gamma\,|0\rangle\,|\psi_{\boldsymbol{\theta}^{(t)}}\rangle - \frac{\mathrm{i}}{C_D}\sin\gamma\,|1\rangle\,D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle\right),$$

where $p_D = \cos^2\gamma + \frac{1}{C_D}\sin^2\gamma\,\langle\psi_{\boldsymbol{\theta}}|D^T D|\psi_{\boldsymbol{\theta}}\rangle$ and C_D is an external parameter that will be explained below. Note that D will be constructed as a real Hermitian operator, and the resulting $D|\psi_{\boldsymbol{\theta}}\rangle$ has

[2] I will use the bold notation when explicitly referring to a classical matrix, and the non-bold symbol when referring the a quantum operator.

real amplitudes. Measure the state in the basis

$$|\text{yes}\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle), \qquad |\text{no}\rangle = \frac{1}{\sqrt{2}}(i|0\rangle + |1\rangle).$$

Measuring 'yes' results in a state

$$|\psi_{\boldsymbol{\theta}^{(t+1)}}\rangle = \frac{1}{\sqrt{2p_D p_{\text{yes}}}}\left(\cos\gamma \;\; |\psi_{\boldsymbol{\theta}^{(t)}}\rangle - \frac{1}{C_D}\sin\gamma \;\; D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle\right),$$

where p_{yes} is the probability of success given by

$$p_{\text{yes}} = \frac{1}{2p_D}\left[\cos^2\gamma + \frac{1}{C_D^2}\sin^2\gamma\langle\psi_{\boldsymbol{\theta}}|D^\dagger D|\psi_{\boldsymbol{\theta}}\rangle - 2\cos\gamma \;\sin\gamma\;\langle\psi_{\boldsymbol{\theta}}|D|\psi_{\boldsymbol{\theta}}\rangle\right].$$

Proof. The state $|\psi\rangle$ in yes-no basis reads

$$\begin{aligned}
&\big(|\text{yes}\rangle\langle\text{yes}| + |\text{no}\rangle\langle\text{no}|\big)\;\frac{1}{\sqrt{p_D}}\big(\cos\gamma|0\rangle|\psi_{\boldsymbol{\theta}^{(t)}}\rangle + \frac{\mathrm{i}}{C_D}\sin\gamma|1\rangle D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle\big)\\
&= \frac{1}{\sqrt{p_D}}\Bigg(\cos\gamma\;\;\langle\text{yes}|0\rangle|\text{yes}\rangle\;|\psi_{\boldsymbol{\theta}^{(t)}}\rangle + \frac{\mathrm{i}}{C_D}\sin\gamma\;\langle\text{yes}|1\rangle|\text{yes}\rangle\;D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle + \cos\gamma\;\;\langle\text{no}|0\rangle\;|\text{no}\rangle|\psi_{\boldsymbol{\theta}^{(t)}}\rangle +\\
&\quad \frac{\mathrm{i}}{C_D}\sin\gamma\;\langle\text{no}|1\rangle|\text{no}\rangle\;D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle\Bigg)\\
&= \underbrace{\frac{1}{\sqrt{2p_D}}\Big(\cos\gamma\;|\psi_{\boldsymbol{\theta}^{(t)}}\rangle + \frac{1}{C_D}\;\sin\gamma\;D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle\Big)}_{|\psi_1\rangle}|\text{yes}\rangle + \underbrace{\frac{1}{\sqrt{2}}\Big(-\,\mathrm{i}\,\cos\gamma\;|\psi_{\boldsymbol{\theta}^{(t)}}\rangle + \mathrm{i}\,\sin\gamma\;\;D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle\Big)}_{|\psi_2\rangle}|\text{no}\rangle
\end{aligned}$$

The probability $|\langle\psi_1|\psi_1\rangle|^2$ of measuring $|\text{yes}\rangle$ is given by

$$p_{\text{yes}} = \frac{1}{2p_D}(\cos^2\gamma + \frac{1}{C_D}\sin^2\gamma\langle\psi_{\boldsymbol{\theta}}|D^\dagger D|\psi_{\boldsymbol{\theta}}\rangle - \cos\gamma\;\sin\gamma\;(\langle\psi_{\boldsymbol{\theta}}|D^\dagger|\psi_{\boldsymbol{\theta}}\rangle + \langle\psi_{\boldsymbol{\theta}}|D|\psi_{\boldsymbol{\theta}}\rangle),$$

which for Hermitian operators can be summarised as above. The probability $|\langle\psi_2|\psi_2\rangle|^2$ of measuring $|\text{no}\rangle$ is given by

$$p_{\text{no}} = \frac{1}{2p_D}(\cos^2\gamma + \frac{1}{C_D}\sin^2\gamma\;\langle\psi_{\boldsymbol{\theta}}|D^\dagger D|\psi_{\boldsymbol{\theta}}\rangle + \cos\gamma\;\sin\gamma\;(\langle\psi_{\boldsymbol{\theta}}|D^\dagger|\psi_{\boldsymbol{\theta}}\rangle + \langle\psi_{\boldsymbol{\theta}}|D|\psi_{\boldsymbol{\theta}}\rangle),$$

which with the definition of p_D adds up to 1. □

Choosing the external parameter γ so that

$$\cos\gamma = \frac{1}{\sqrt{1+\eta^2/C_D^2}}, \qquad \sin\gamma = \frac{\eta}{\sqrt{1+\eta^2/C_D^2}},$$

which results in

$$|\psi_{\boldsymbol{\theta}^{(t+1)}}\rangle = \frac{1}{C^{(t+1)}}\Big(|\psi_{\boldsymbol{\theta}^{(t)}}\rangle - \eta|\nabla o(\boldsymbol{\theta}^{(t)})\rangle\Big),$$

where the normalisation term is given by

$$C^{(t+1)} = 1 - 2\eta\langle\psi_{\boldsymbol{\theta}^{(t)}}|D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle + \eta\;\langle\psi_{\boldsymbol{\theta}^{(t)}}|D^2|\psi_{\boldsymbol{\theta}^{(t)}}\rangle.$$

This encodes the classical vector

$$\boldsymbol{\theta}^{(t+1)} = \frac{1}{C^{(t+1)}}(\theta^{(t)} - \eta(\nabla o(\boldsymbol{\theta}^{(t)}),$$

which is exactly a gradient descent update which is normalised to unit length. With the choice of the free parameter γ, and noting that $\langle\psi_{\boldsymbol{\theta}^{(t)}}|D|\psi_{\boldsymbol{\theta}^{(t)}}\rangle$ is the inner product of the current state with the gradient and $\langle\psi_{\boldsymbol{\theta}^{(t)}}|D^2|\psi_{\boldsymbol{\theta}^{(t)}}\rangle$ the length of the gradient, the acceptance probability can also be written as

$$p_{\text{yes}} = \frac{1}{2} - \frac{2\eta\ (\boldsymbol{\theta}^{(t)})^T\ \nabla o(\boldsymbol{\theta}^{(t)})}{1 + \eta^2\ \nabla o(\boldsymbol{\theta}^{(t)})^T\ \nabla o(\boldsymbol{\theta}^{(t)})}.$$

Series expansion of this expression reveals

$$p_{\text{yes}} = \frac{1}{2} - 2\eta(\boldsymbol{\theta}^{(t)})^T\ \nabla o(\boldsymbol{\theta}^{(t)}) + \mathcal{O}(\eta^2),$$

which is sufficiently large for small enough η.

Note that the cost for one step in the quantum gradient descent method depends on the time to implement and apply the operator $\mathbf{D}$, and the preparation for copies of the initial state. If no specific guess for the initial state is given, it can be chosen conveniently so that time is at worst linear in the number of qubits or $\mathcal{O}(logN)$. Each update step furthermore has a probability of $1 - p_{\text{yes}}$ to fail.

8.2.2 How to calculate the gradient in amplitude encoding

The following shows how to implement the operator $D|\psi_{\boldsymbol{\theta}}\rangle = |\psi_{\nabla}\rangle$ used for the quantum gradient descent step above. The index t indicating the current step is omitted for readability, and ∇ is a shorthand for $\nabla o(\boldsymbol{\theta}^{(t)})$ (the gradient of the objective function is always evaluated at the current point).

As explained before, to implement $D|\psi_{\boldsymbol{\theta}}\rangle$ one can evolve the quantum system with a Hamiltonian corresponding to D, $e^{iD\Delta t}|\psi_{\boldsymbol{\theta}}\rangle$, and use the techniques from [90] to write D's eigenvalues into the amplitudes. The problem is that we cannot guarantee D to be sparse in matrix representation, and again the solution is to resort to the density matrix exponentiation technique, however with a novel variation. Instead of simulating a swap operator, a s-sparse operator M_D which is specifically constructed for the purpose will be used to exponentiate D:

$$M_D = \frac{1}{2}\sum_j \sum_\alpha \left(\bigotimes_{i\neq j} A_i^\alpha\right) \otimes \left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right).$$

This operator is similar to Equation (8.4), but sums over different permutations of the A_j^α. In the j'th term, the operator acting on the last Hilbert space is given by the matrix $\mathbf{A}_j^\alpha$.

Example 8.2.1: The operator M_D

For $p = 2$, we get

$$M_D = \sum_\alpha (A_1^\alpha \otimes A_2^\alpha + A_2^\alpha \otimes A_1^\alpha).$$

In each term of the sum, another A_i^α, $i = 1...2$ acts on the 2nd Hilbert space.

Applying this operator to p copies of the quantum system ρ and taking the trace over all but the last system approximates the exponentiation of D with error Δt^2,

$$\mathrm{tr}_{p-1}\{e^{-iM_D\Delta t}\ \rho^{\otimes p}\ e^{iM_D\Delta t}\} = e^{-iD\Delta t}\rho\, e^{iD\Delta t} + \mathcal{O}(\Delta t^2).$$

Proof. The operator D from Equation 8.6 which we seek to implement contains the factors $\theta^\dagger A_i^\alpha \theta$. If $\boldsymbol{\theta}$ is interpreted as a quantum state vector and A_i^α an observable, this corresponds to the expectation value $\langle\psi_{\boldsymbol{\theta}}|A_i^\alpha|\psi_{\boldsymbol{\theta}}\rangle = \mathrm{tr}\{A_i^\alpha\rho\}$ where $\rho = |\psi_{\boldsymbol{\theta}}\rangle\langle\psi_{\boldsymbol{\theta}}|$ is the corresponding density matrix. The full operator $D = \sum_j \sum_\alpha \left(\prod_{i\neq j}\theta^\dagger A_i^\alpha\theta\right)\left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right)$ can therefore be reproduced by a quantum operator that is formally equal to

$$D = \mathrm{tr}_{1\ldots p-1}\{\rho^{\otimes(p-1)}\ M_D\}$$

where $\rho^{\otimes(p-1)} = \rho\otimes\ldots\otimes\rho\otimes\mathcal{I}$ is the joint quantum state of $p-1$ copies of ρ and an identity. To show this, consider the relation

$$\begin{aligned}
D &= \frac{1}{2}\sum_{\alpha,j}\left(\prod_{i\neq j}\langle x|A_i^\alpha|x\rangle\right)\left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right)\\
&= \frac{1}{2}\sum_{\alpha,j}\left(\prod_{i\neq j}\mathrm{tr}\{\rho A_i^\alpha\}\right)\left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right)\\
&= \frac{1}{2}\sum_{\alpha,j}\mathrm{tr}\left\{\underbrace{(\rho\otimes\ldots\otimes\rho)}_{p-1\text{ times}}\left(\bigotimes_{i\neq j}A_i^\alpha\right)\right\}\left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right)\\
&= \frac{1}{2}\sum_{\alpha,j}\mathrm{tr}_{p-1}\left\{(\underbrace{\rho\otimes\ldots\otimes\rho}_{p-1\text{ times}}\otimes\mathcal{I})\left[\left(\bigotimes_{i\neq j}A_i^\alpha\right)\otimes\left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right)\right]\right\}\\
&= \frac{1}{2}\mathrm{tr}_{p-1}\left\{(\underbrace{\rho\otimes\ldots\otimes\rho}_{p-1\text{ times}}\otimes\mathcal{I})\left[\sum_{\alpha,j}\left(\bigotimes_{i\neq j}A_i^\alpha\right)\otimes\left((A_j^{(\alpha)})^T + A_j^{(\alpha)}\right)\right]\right\}\\
&= \mathrm{tr}_{p-1}\left\{(\underbrace{\rho\otimes\ldots\otimes\rho}_{p-1\text{ times}}\otimes\mathcal{I})M_D\right\}
\end{aligned}$$

With this formal equality one can show that exponentiating M_D applied to p copies of the state ρ (joint with one identity) approximates the procedure of exponentiating D and applying it to one copy of ρ.

$$\begin{aligned}
\mathrm{tr}_{p-1}\{e^{-iM_D\Delta t}\ \rho^{\otimes p}\ e^{iM_D\Delta t}\} &= \mathrm{tr}_{p-1}\{\rho^{\otimes p} + i\Delta t[\rho^{\otimes p}, M_D]\} + O(\Delta t^2)\\
&= \rho + i\Delta t\ \mathrm{tr}_{p-1}\{[\rho^{\otimes p}, M_D]\} + O(\Delta t^2)\\
&= \rho + i\Delta t\ \rho\mathrm{tr}_{p-1}\{(\rho^{\otimes p-1}\otimes\mathcal{I})M_D\} - i\Delta t\ \mathrm{tr}_{p-1}\{M_D(\rho^{\otimes p-1}\otimes\mathcal{I})\}\rho + O(\Delta t^2)\\
&= \rho + i\Delta t\ \rho\mathrm{tr}_{p-1}\{(\rho^{\otimes p-1}\otimes\mathcal{I})M_D\} - i\Delta t\ \mathrm{tr}_{p-1}\{(\rho^{\otimes p-1}\otimes\mathcal{I})M_D\}\rho + O(\Delta t^2)\\
&= \rho + i\Delta t\ \rho D - i\Delta t\ D\rho + O(\Delta t^2)\\
&\approx e^{-iD\Delta t}\rho\, e^{iD\Delta t}
\end{aligned}$$

□

Note that in general M_D might be a non-Hermitian operator, but as mentioned before we can consider the Hermitian extended operator

$$\hat{M}_D \coloneqq \begin{bmatrix} 0 & M_D \\ M_D^\dagger & 0 \end{bmatrix}, \tag{8.9}$$

For the following discussion it is therefore assumed that without loss of generality M_D is Hermitian.

As explained in the last chapter and shown in Appendix A.2, we have to extend this procedure in order to prepare the state for the quantum phase estimation, for which another $\mathcal{O}(\epsilon^{-3})$ copies of each of the p Hilbert spaces is needed. Afterwards, quantum phase estimation and a postselective amplitude update complete the matrix multiplication as described repeatedly in earlier chapters. In order to use this technique within the framework of an update step in the last section, the procedure has to be executed controlled by the ancilla in Equation 8.8. I will briefly sketch the particulars of integrating density matrix exponentiation with the updating step.

Let $\{|\chi_l\rangle\}$ be an eigenbasis of D with corresponding eigenvalues $\{\lambda_l\}$ so that the current state can be written as $|\psi_{\boldsymbol{\theta}}\rangle = \sum_l \beta_l |\chi_l\rangle$ with $\beta_l = \langle \chi_l || \psi_{\boldsymbol{\theta}} \rangle\rangle$. At the beginning of each update we have copies of the state

$$(\cos\gamma\,|0\rangle + \mathrm{i}\,\sin\gamma\,|1\rangle)\,|\psi_{\boldsymbol{\theta}}\rangle. \tag{8.10}$$

After the density matrix exponentiation and quantum phase estimation conditioned on the $|1\rangle$ branch of the ancilla, we obtain:

$$|\psi\rangle = \left(\cos\gamma\,|0\rangle|\psi_{\boldsymbol{\theta}}\rangle|0\rangle + \mathrm{i}\,\sin\gamma\,|1\rangle \sum_l \beta_l |\chi_l\rangle|\lambda_l\rangle\right), \tag{8.11}$$

Now perform the postselective amplitude update and uncompute the eigenvalue register in the $|1\rangle$-branch to arrive at the state

$$\frac{1}{\sqrt{p_D}}\left(\cos\gamma\,|0\rangle|\psi_{\boldsymbol{\theta}}\rangle + \mathrm{i}\,\sin\gamma\,|1\rangle \sum_l C_D \lambda_l \beta_l |\chi_l\rangle|\lambda_l\rangle\right). \tag{8.12}$$

We chose a constant $C_D = \mathcal{O}(1/\kappa_D)$, where κ_D is the condition number of D. This result is equivalent to the desired state

$$\frac{1}{\sqrt{p_D}}\left(\cos\gamma\,|0\rangle|\psi_{\boldsymbol{\theta}}\rangle + \mathrm{i}\,C_D \sin\gamma\,|1\rangle D|\psi_{\boldsymbol{\theta}}\rangle\right). \tag{8.13}$$

The success probability of the measurement is given by

$$p_D = \cos^2\gamma + C_D^2 \sin^2\gamma\,\langle\psi_{\boldsymbol{\theta}}|D^2|\psi_{\boldsymbol{\theta}}\rangle. \tag{8.14}$$

8.2.3 Resources needed for quantum gradient descent

The number of operations in each iteration of quantum gradient descent is defined by different subroutines:

- the multiplication with D needs of the order of $\mathcal{O}(\frac{1}{p_D})$ repetitions to be successful
- the update step needs of the order of $\mathcal{O}(\frac{1}{p_{\text{yes}}})$ repetitions to be successful
- density matrix exponentiation requires a number of operations that is polynomial in $p, s, \log N$

While the logarithmic dependency on the dimension in the number of operations is the main advantage of the quantum method, its main caveat lies in the number of copies of the current state required to produce a successful copy for the next iteration. Since some operations in the algorithm are only successful to a certain probability, one requires a large number of copies to make sure that at least some quantum systems have performed the computation successfully. On top of this, the accuracy with which density matrix exponentiation is performed also grows with the number of copies used. For example, if in every iteration half of the copies are consumed, the number of systems that need to be prepared in the initial state grows exponentially with the number of iterations T. This point seems intrinsic to quantum iterative methods and requires further investigation.

The number of copies that are on average 'consumed' in one iteration of the method can be estimated as follows: Following a more refined error analysis for density matrix exponentiation with erroneous inputs presented (see Appendix B.3) we require of the order of $\mathcal{O}(\frac{1}{p_D p_{\text{yes}}}\left(1+\frac{p^2\epsilon_t^2}{\epsilon_{t+1}}\right)\frac{1}{\epsilon_{t+1}^2})$ copies of the input (accurate up to error ϵ_t) to the tth iteration to gain one copy with utmost error ϵ_{t+1}.

8.3 Extension to Newton's method

A quantum algorithm for Newton's method follows the same scheme as the quantum gradient descent method. The only difference is that after the map $D|\psi_{\boldsymbol{\theta}}\rangle = |\psi_\nabla\rangle$ one also has to also apply an operator H^{-1} to $|\nabla\rangle$ which implements the inverse Hessian matrix. This can be done in the same manner as the implementation of D, but replacing the quantum matrix multiplication by a matrix inversion routine as outlined in Section 3.4.5. This shall be summarised here, while the details can be found in Appendix B.3.

The Hessian in Equation (8.7) can be decomposed into

$$\mathbf{H} = \mathbf{H}_A + \mathbf{D}, \tag{8.15}$$

with $\mathbf{D}$ as above. To obtain the eigenvalues of $\mathbf{H}$ via phase estimation, we exponentiate a total operator H by exponentiating the operators H_A and D sequentially using the standard Lie product formula

$$\mathrm{e}^{\mathrm{i}\,H\Delta t} \approx \mathrm{e}^{\mathrm{i}\,H_A\Delta t}\mathrm{e}^{\mathrm{i}\,D\Delta t} + \mathcal{O}(\Delta t^2). \tag{8.16}$$

To implement the individual exponentiations themselves we use a similar trick as before. We associate a simulable operator M_{H_A} with H_A and the operator M_D with D as before. (This rather technical definition can be found in Appendix B.3). The operator M_{H_A} is sparse if the constituent matrices A_j^α are sparse and can be constructed if they are given in oracular form. Denote by s' the sparsity of M_{H_A}. Let σ be an arbitrary state on which the matrix exponential shall be applied

on, we can use multiple copies of the current state $\rho = |\psi_\theta\rangle\langle\psi_\theta|$ to perform

$$\begin{aligned} &\mathrm{tr}_{1\dots p-1}\{\mathrm{e}^{-\mathrm{i}\, M_{H_A}\Delta t}\,(\rho\otimes\ldots\otimes\rho)\otimes\sigma\,\mathrm{e}^{\mathrm{i}\, M_{H_A}\Delta t}\} \\ &\qquad\approx \mathrm{e}^{-\mathrm{i}\, H_A\Delta t}\,\sigma\,\mathrm{e}^{\mathrm{i}\, H_A\Delta t}. \end{aligned} \tag{8.17}$$

The Lie product formula adds an error of the order $\mathcal{O}(\Delta t^2)$ that has to be considered in the analysis.

The application of the inverse Hessian for Newton's method runs polynomial in the sparsity s' and only increases the number of required copies by a factor of order $\mathcal{O}\left(\frac{1}{p_H}\left(1+\frac{p^2\epsilon_t^2}{\epsilon_{t+1}}\right)\frac{1}{\epsilon_{t+1}^3}\right)$, where p_H is the equivalent to p_D but for the multiplication of the inverse Hessian.

In summary, the runtime reflects the exponential speed-up in the size of the inputs (here the parameter space) known from other quantum machine learning algorithms, provided the A_i^α are sparse and accessible via an oracle for simulation. On the downside, the share of copies consumed in every step restricts us to only perform very few iterations. This is unfortunate for general training procedures which require many iterations when searching for the optimal parameter, but may prove useful in cases where the initial guess is close to the desired optimum. An example are situations in which the parameters have been pre-trained by other methods. This appears typically in deep neural networks, where restricted Boltzmann machines or autoencoders prepare the parameter vector. Another suitable situation arises when the number of the parameters is large enough to justify the spatial resources needed for the quantum algorithm. Fast convergence in the proximity of optima is guaranteed for Newton's method.

This study has only looked at a very specific type of objective functions, namely homogeneous polynomials of even order with unit sphere constraints. As mentioned earlier, for applications in machine learning other objective functions, as well as the effect of normalisation constraints as a possible regularisation mechanism need to be investigated further.

Chapter 9

Quantum nearest neighbour algorithm

The first part of Section 9.1 (introducing the quantum nearest neighbour algorithm in basis encoding) has been published in Ref. [225]: Schuld, Sinayskiy, Petruccione (2014) Quantum Computing for Pattern Classification, Lecture Notes in Computer Science Vol 8862, Springer Verlag, pp. 208-220. I was responsible for the idea, development, analysis and write-up of the content of the publication. The second part (the quantum nearest neighbour algorithm in amplitude encoding) has been further developed together with Mark Fingerhut and the manuscript "Computing distances with quantum interference: An implementation of a weighed nearest neighbour quantum machine learning algorithm" is about to be submitted.

The k-nearest neighbour algorithm has been introduced in Section 2.3.4.2 of Chapter 2 as one of the most simple but yet surprisingly successful classifiers. It can be formulated as the following rule:

Predict the class for a new input that most of its k closest training vectors are assigned to.

Quantum algorithms that apply Grover search to k nearest neighbour have been discussed in the literature review of Section 6.2. It has also been mentioned that some versions of the method weigh the neighbours by their distance to the new input, so that closer neighbours have more influence on the prediction than those further away. With this adaptation one takes into account the entire dataset of M inputs, which effectively means that $k = M$. The weighing rule then defines how fast the 'influence' decreases with distance and can therefore be understood as a distance measure or kernel $\kappa(|\tilde{\mathbf{x}} - \mathbf{x}^m|)$. An illustration is presented in Figure 9.1.

In this chapter I will consider a binary pattern classification problem and present two versions of a novel quantum nearest neighbour algorithm, one which represents the dataset in basis encoding and the other representing the training inputs in amplitude encoding as well as the target outputs in basis encoding (i.e., by a single qubit). In both cases the idea is to construct a quantum state in which the probability of measuring one of M basis states corresponds to the weight assigned to the corresponding input, which in turn is a function of its distance to the new input. In other words, the amplitude distribution takes the role of the kernel function. A measurement of the output qubit then reveals the probability of the class being represented by the training inputs in close

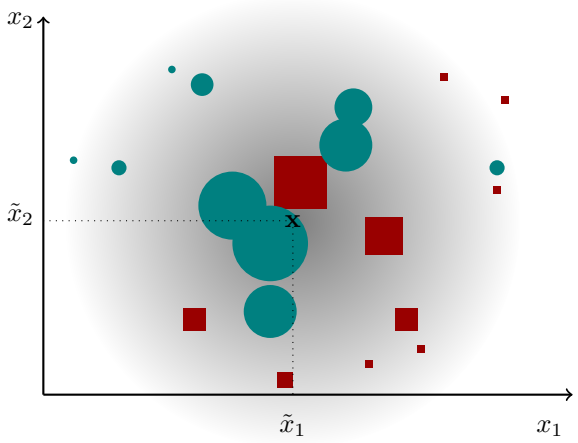


Figure 9.1: Illustration of all-nearest neighbour where the neighbours are weighted by the Euclidean distance to the new input. The symbols show the 2-dimensional inputs that have each a class attribute 'circle' or 'rectangle'. The new input is located at $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2)$, and as in the k-nearest neighbour illustration it will be classified as a circle.

proximity to the new input. Although not necessarily introducing a speed-up compared to the quantum method, the quantum machine learning algorithms are a fruitful demonstration of how to represent a distance measure in the amplitudes and are pre-designed for a comparably simple implementation in current-day hardware.

9.1 Quantum nearest neighbour routine in basis encoding

The idea of the quantum algorithm in basis encoding is to create a superposition in which each basis state corresponds to a training point, and to write the square root of the weighing factor (or the value of the kernel function) into the corresponding amplitude,

$$\sum_{m=1}^{M} \sqrt{\kappa(|\tilde{\mathbf{x}} - \mathbf{x}^m|)} |\mathbf{x}^m, y^m\rangle. \tag{9.1}$$

For this a simple routine developed by Trugenberger [226] will be applied, which leads to a cosine kernel (restricted to the interval $[0, \pi]$).

The data is represented by the basis states $|\mathbf{x}^m, y^m\rangle$ (for details, see Section 3.3.1). I assume for simplicity that the features as well as the class are binary, and choose the distance measure to be the Hamming distance. The m'th basis state in the superposition then becomes $|x_1^m, ..., x_N^m; y^m\rangle$. The algorithm can easily be adapted to the case of continuous numbers and multi-label classification, with the only difference in the routine to extract the distance measure.

Doing a measurement on the last qubit $|y^m\rangle$ of the superposition reveals the combined weight of each class. The measurement outcome will be 0 with a probability

$$\sum_{m|y^m=-1} \kappa(|\tilde{\mathbf{x}} - \mathbf{x}^m|),$$

and 1 with probability

$$\sum_{m|y^m=1} \kappa(|\tilde{\mathbf{x}} - \mathbf{x}^m|).$$

Alternatively, one can turn the algorithm into a 'stochastic' or 'sampling' version of k-nearest-neighbour by measuring the entire state repeatedly, effectively taking samples from the superposition. Training inputs that are closer will have a higher probability to be the outcome, and the state of their class qubit is recorded.

Of course, in both versions the measurement destroys the state and for each classification it has to be re-prepared, which requires to store and access the entire dataset like the classical method does. The runtime of the routine depends on the resources needed for state preparation, which is in general linear in M, N and therefore similar to classical k-nearest neighbour. If faster state preparation is available or the dataset superposition is the result of a previous quantum computation, the routine is independent of M by applying the weighing step fully in parallel (although in the $k < M$ case the number of samples required for a certain accuracy may depend on M).

To start with, a state preparation procedure (such as presented in Section 4.1) is used to encode the training set into a uniform superposition $1/\sqrt{M}\sum_{m=1}^{M}|x_1^m, ..., x_N^m; y^m\rangle$ joined with a register encoding the new input as well as an ancilla in superposition:

$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M}|x_1, ..., x_N\ ;\ y^m\rangle|\tilde{x}_1, ..., \tilde{x}_N\rangle(|0\rangle + |1\rangle).$$

Here and unless stated different, global normalisation constants will be ignored since the desired result is a relative value. An XOR gate applied to every qubit x_j and $\tilde{x}_j$ for $j = 1...N$ compares the two vectors and writes the result

$$d_j^m = \begin{cases} 1, & \text{if } x_j^m = \tilde{x}_j, \\ 0, & \text{else,} \end{cases}$$

into the register previously containing the new input,

$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M}|x_1, ..., x_N\ ;\ y^m\rangle|d_1^m, ..., d_N^m\rangle(|0\rangle + |1\rangle).$$

A unitary $e^{-i\frac{\pi}{2N}H}$ with the Hamiltonian

$$H = 1 \otimes 1 \otimes \sum_j \left(\frac{(\sigma_z)_j + 1}{2}\right) \otimes \sigma_z,$$

which $(\sigma_z)_j$ acting on d_j, has the effect of writing the sum of the d_j, in other words the Hamming distance, into the phase with a sign conditioned on the last ancilla qubit,

$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M} e^{i\frac{\pi}{2N}\sum_j d_j^m}|x_1, ..., x_N\ ;\ y^m\rangle|d_1^m, ..., d_N^m\rangle|0\rangle + \frac{1}{\sqrt{M}}\sum_{m=1}^{M} e^{-i\frac{\pi}{2N}\sum_j d_j^m}|x_1, ..., x_N\ ;\ y^m\rangle|d_1^m, ..., d_N^m\rangle|1\rangle.$$

A Hadamard gate on the last ancilla qubit then interferes the two terms and results in

$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M}\cos(\frac{\pi}{2N}\sum_j d_j^m)|x_1,...,x_N\ ;\ y^m\rangle|d_1^m,...,d_N^m\rangle|0\rangle+$$
$$\frac{1}{\sqrt{M}}\sum_{m=1}^{M}\sin(\frac{\pi}{2N}\sum_j d_j^m)|x_1,...,x_N\ ;\ y^m\rangle|d_1^m,...,d_N^m\rangle|1\rangle.$$

A conditional measurement on the ancilla selects the cosine branch. The probability of acceptance is given by

$$p_{\text{acc}} = \frac{1}{M}\sum_{m=1}^{M}\cos^2\left[\frac{\pi}{2N}\sum_j d_j^m\right].$$

This probability is also a measure of how close the data is to the new input: If the collective Hamming distance is large, the sine branch of the superposition will have a larger probability to be measured. In the worst case scenario, all training vectors have a Hamming distance close to N and the probability of the conditional measurement to succeed will be close to zero. However, in this case the data might not reveal a lot of information for the classification of the new input anyways, and the probability of acceptance can therefore be seen as a measure of how well-posed the classification problem is in the first place.

After a successful conditional measurement, the state becomes proportional to

$$\sum_{m=1}^{M}\cos(\frac{\pi}{2N}\sum_j d_j^m)|x_1^m,...,x_N^m\ ;\ y^m\rangle,$$

where the $|d_1,...,d_N\rangle|0\rangle$ registers were ommitted. This is precisely the desired outcome defined in Equation (9.1). The kernel function used is a cosine, and the factor of $\frac{\pi}{2N}$ normalises the argument in the cosine to the interval $[0,\pi]$ where it is a monotonically decreasing function, similar to a Gaussian. As required, training vectors with a larger Hamming distance to the new input get a smaller weight than those closer to it.

The two versions of the nearest neighbour algorithm can now be implemented by choosing the measurement. A measurement on the class qubit y^m will have a probability of

$$p(\tilde{y}=-1) = p(|y^m\rangle = |0\rangle) = \sum_{m|y^m=0}\cos^2(\frac{\pi}{2N}\sum_j d_j^m),$$

to predict the class -1 and a complementary probability of

$$p(\tilde{y}=1) = p(|y^m\rangle = |1\rangle) = \sum_{m|y^m=1}\cos^2(\frac{\pi}{2N}\sum_j d_j^m),$$

to predict class 1. Alternatively, measuring the entire basis state $|x_1^m,...,x_N^m\ ;\ y^m\rangle$ has a probability of

$$p(\mathbf{x}^m) = \cos^2(\frac{\pi}{2N}\sum_j d_j^m),$$

to pick the mth training vector, and closer training vectors are thus preferred. Doing this

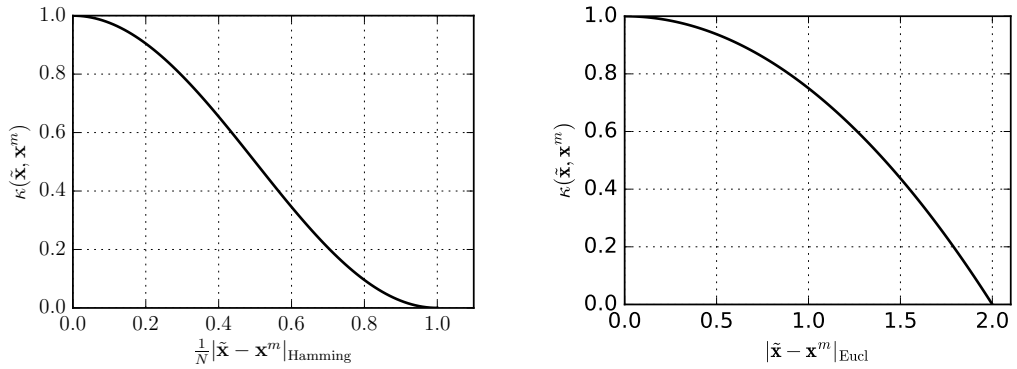


Figure 9.2: The two kernel functions of the quantum nearest neighbour algorithm. In basis encoding a cosine kernel is used (left) while in amplitude encoding a square kernel appears (right).

repeatedly can be understood as a stochastic procedure to select the closest vectors and to assign the new class via majority vote.

Representing the dataset in basis encoding is rather expensive in terms of spatial resources, especially when dealing with continuous features: The number of qubits needed if every real entry of the input vector is encoded in τ qubits is of the order $N\tau$. For example, if the features are encoded as binary fractions, a precision of τ would imply an error of less than $\frac{1}{2^\tau}$. To have an error of less than 0.01, we need at least $\tau = 7$ qubits per feature. Considering the technological challenges to scale quantum computing to a large number of qubits, this is quickly prohibitive for large input spaces. It is therefore interesting to develop a similar routine based on the much more compact amplitude encoding.

9.2 Quantum nearest neighbour routine in amplitude encoding

For amplitude encoding we can relax the assumption of binary features and consider continuous features for a binary classification problem. For this one requires the (normalised) data set and new input to be encoded in the amplitudes of a quantum state of the form

$$\frac{1}{\sqrt{2M}} \sum_{m=1}^{M} \left(|0\rangle|\psi_{\tilde{\mathbf{x}}}\rangle + |1\rangle|\psi_{\mathbf{x}^m}\rangle\right)|y^m\rangle|m\rangle,$$

with

$$|\psi_{\mathbf{x}^m}\rangle = \sum_{i=1}^{N} x_i^m|i\rangle, \quad |\psi_{\tilde{\mathbf{x}}}\rangle = \sum_{i=1}^{N} \tilde{x}_i|i\rangle.$$

(In vector notation this state is a concatenation of the vectors $\tilde{\mathbf{x}}, \mathbf{x}^1, \tilde{\mathbf{x}}, \mathbf{x}^2, ...\tilde{\mathbf{x}}, \mathbf{x}^M$.) A Hadamard gate on the ancilla interferes the two states and results in

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^{M} \Big(|0\rangle\left[|\psi_{\tilde{\mathbf{x}}}\rangle + |\psi_{\mathbf{x}^m}\rangle\right] + |1\rangle\left[|\psi_{\tilde{\mathbf{x}}}\rangle - |\psi_{\mathbf{x}^m}\rangle\right]\Big)|y^m\rangle|m\rangle.$$

A conditional measurement to find the ancilla in $|0\rangle$ succeeds with probability $p_{\text{acc}} = \frac{1}{4M}\sum_m \sum_i |\tilde{x}_i + x_i^{(m)}|^2$ which is equal to $1 - \frac{1}{4M}\sum_m \sum_i |\tilde{x}_i - x_i^{(m)}|^2$, i.e. it is more likely to succeed if the collective Euclidean distance of the training set to the new input is small. This is similar to the conditional measurement in the basis encoding routine. In the worst case, $\tilde{x}_i \approx -x_i^{(m)}$ for all $m = 1...M$, and acceptance will be very unlikely. However, this means also here that the new input is 'far away' from the dataset, an indicator for the low expressive power of a classification algorithm based on distances.

If the conditional measurement was successful, the result is proportional to

$$\frac{1}{2\sqrt{M}} \sum_{m=1}^{M} \sum_{i=1}^{N} \left(\tilde{x}_i + x_i^{(m)}\right) |i\rangle |y^{(m)}\rangle |m\rangle.$$

The probability of measuring the class qubit $|y^{(m)}\rangle$ in state 0 and predict class -1 is given by

$$p(\tilde{y} = -1) = \frac{1}{4M} \sum_{m|y^m=0} |\tilde{\mathbf{x}} + \mathbf{x}^{(m)}|^2 = \frac{1}{M} \sum_{m|y^m=0} 1 - \frac{1}{4}|\tilde{\mathbf{x}} - \mathbf{x}^{(m)}|^2.$$

Proof. The last equality has been already used above and is easy to show:

$$\begin{aligned}
\frac{1}{4}|\tilde{\mathbf{x}} + \mathbf{x}^{(m)}|^2 &= \frac{1}{4}\sum_{i=1}^{N} |\tilde{x}_i + x_i^{(m)}|^2 \\
&= \frac{1}{4}\sum_{i=1}^{N} |\tilde{x}_i|^2 + \frac{1}{2}\sum_{i=1}^{N} (\tilde{x}_i)^* x_i^{(m)} + \frac{1}{2}\sum_{i=1}^{N} \tilde{x}_i (x_i^{(m)})^* + \frac{1}{4}\sum_{i=1}^{N} |x_i^{(m)}|^2 \\
&= \frac{1}{2} + \frac{1}{2}(\tilde{x}_i)^* x_i^{(m)} + \frac{1}{2}\tilde{x}_i (x_i^{(m)})^* \\
1 - \frac{1}{4}|\tilde{\mathbf{x}} - \mathbf{x}^{(m)}|^2 &= 1 - \frac{1}{4}\sum_{i=1}^{N} |\tilde{x}_i - x_i^{(m)}|^2 \\
&= 1 - \frac{1}{4}\sum_{i=1}^{N} |\tilde{x}_i|^2 + \frac{1}{2}\sum_{i=1}^{N} (\tilde{x}_i)^* x_i^{(m)} + \frac{1}{2}\sum_{i=1}^{N} \tilde{x}_i (x_i^{(m)})^* - \frac{1}{4}\sum_{i=1}^{N} |x_i^{(m)}|^2 \\
&= \frac{1}{2} + \frac{1}{2}(\tilde{x}_i)^* x_i^{(m)} + \frac{1}{2}\tilde{x}_i (x_i^{(m)})^*
\end{aligned}$$

□

Expressing the probability to predict class -1 by the squared distance shows that it is higher the closer the class -1 training vectors are to the input. The kernel is hence given by $\kappa(\tilde{\mathbf{x}} - \mathbf{x}^{(m)}) = 1 - \frac{1}{4M}|\tilde{\mathbf{x}} - \mathbf{x}^{(m)}|^2$, and is therefore a polynomial kernel. Note that also in this case, as an alternative the m-register could be measured to extract training inputs and their classes with a probability depending on their squared distance to the new input.

Chapter 10

Implementing a perceptron on a quantum computer

The content of Section 10.3 of this chapter has been published in Ref [227]: Schuld, Sinayskiy, Petruccione (2015) Simulating a perceptron on a quantum computer. Physics Letters A 379.7: 660-663. I was responsible for the idea, development, analysis and write-up of the content of the publication.

When looking for literature on *quantum neural networks* or *quantum perceptrons* one will find hundreds of models that claim to combine quantum mechanics and neural networks in one way or the other, initially with the goal of finding a "quantum brain model" and then increasingly targeting the use of quantum information for pattern recognition. These models largely remain in a niche of the quantum information literature due to their failure to recover the essence of neural networks, which is the concatenation of linear and nonlinear updates for the classifier, and a strategy to solve the nonconvex optimisation problem of backpropagation or similar methods. Especially the latter remains an open challenge as it means to solve a class of complicated optimisation problems, which has been the subject of previous chapters. Many proposals therefore propose mixed strategies, such as classical training of a quantum classifier.

During the course of my PhD research I have conducted many attempts to develop a quantum algorithm that recovers (and speeds up) the classification and training step of neural networks, but with at most mixed results. One of these results is an elegant implementation of the step activation function for basis-encoded inputs which will be presented in the last section of this chapter. Before presenting this idea, I want to give a systematic account of the problems when attempting to design quantum algorithms for neural networks. This account will use the approach of distinguishing between different types of information encoding. In order to arrive at a working definition for quantum neural network models I will follow the these design principles:

1. The model has to be implementable on a universal quantum computer, i.e. we can formulate the model as a quantum circuit.

2. The model defines an input-output mapping from either $\mathbb{R}^N$ or $\{0,1\}^N$ to $\mathbb{R}$ or $\{0,1\}$ in order to solve problems of supervised pattern recognition.

3. The mapping consists of a concatenation of linear *and nonlinear* operations.

4. Input and output information have to be encoded in the same form so that multi-layer architectures can be built

5. A training procedure has to be given. Of special interest are coherent training algorithms that make use of the advantages of quantum information processing.

6. The model has to demonstrate a qualitative difference compared to classical neural networks.

Particularly the first, fifth and sixth point are often missed by authors that claim that they developed a quantum neural network model.

10.1 The quantum perceptron as a toy model

To illustrate some of the challenges in developing quantum neural networks under the principles outlined above, it helps to consider a minimal example. A good point of leverage is the building block of neural networks, the perceptron model (see Figure 2.13). Once a perceptron model based on the design principles is given, it can be used to build more complex structures such as feed-forward or recurrent neural networks. As a reminder, the most general form of the input-output relation of a perceptron reads

$$y = \varphi(\mathbf{w}^T\mathbf{x}),$$

where the weight vector $\mathbf{w} \in \mathbb{R}^N$ contains the parameters and $\mathbf{x} \in \mathbb{R}^N$ is the input vector. As before, it will be assumed that the bias term (terms of zero'th order in $\mathbf{w}$) is included in the vectors by setting $x_0 = 1$. The nonlinear map $\varphi : \mathbb{R}^N \to \mathbb{R}$ stands for an activation function that has to be further specified.

Design Principle 1 suggests that the perceptron model can be represented by a general n-qubit system,

$$|\psi\rangle = \sum_{i=0}^{N-1} a_i|i\rangle, \qquad a_i \in \mathbb{C}, \sum_i |a_i|^2 = 1, \tag{10.1}$$

that evolves according to the laws of quantum mechanics. The first question to be answered when designing a quantum version of a perceptron is how the inputs/outputs as well as the weights are encoded into the n-qubit quantum system. This decision determines the tools and limitations given by quantum theory in designing the forward computation (the classification step) as well as the training procedure. Recall the three basic information encoding strategies from Section 3.3: Basis encoding, amplitude encoding and dynamic encoding. Table 10.1 summarises the main resulting options that arise for quantum neural network models. They can be distinguished by approaches that encode the inputs/outputs either in the basis states or the amplitudes. Not listed is the option of encoding the inputs/outputs into a unitary operator, since this means that the outputs of the algorithm has to be a unitary as well, which would be a quantum algorithm that maps a quantum evolution to a quantum evolution, something which is not well defined. Furthermore, the cases where weights are amplitude encoded and inputs/outputs are basis encoded or vice versa do not allow for an obvious strategy of computing the inner product $\mathbf{w}^T\mathbf{x}$. Of course, this does not exclude that such strategies can lead to interesting results.

Encoding			Implementation	
Formalism	**Inputs**	**Weights**	**Training**	**Challenge**
$\|x_1...x_N\rangle, \|w_1...w_N\rangle$	basis	basis	quantum search over all possible $\|w_1...w_N\rangle$	spatial resources
$U_{\mathbf{w}}\|x_1...x_N\rangle$	basis	unitary	quantum feedback or classical optimisation	coherent training scheme
$U_{\mathbf{w}} \sum_i x_i \|i\rangle$	amplitude	unitary		
$\sum_i x_i \|i\rangle, \sum_j w_j \|j\rangle$	amplitude	amplitude	search over amplitude distributions	nonlinearity

Table 10.1: Overview of the options to encode the inputs and outputs as well as the weight parameters of a simple perceptron model into a n-qubit quantum state. The last columns names a central challenge in quantum neural network design, namely the excessive amount of spatial resources needed for Case 1, the design of a coherent training scheme for Cases 2 and 3 as well as the implementation of the nonlinear activation function for Case 4.

10.2 Representing the neurons in amplitude encoding

Let us first have a look at approaches that encode inputs and outputs into the amplitudes of quantum states. Since amplitudes have continuous values we can consider a perceptron model mapping from $\mathbb{R}^N$ to the $\mathbb{R}$. We have to realise two steps. The first step is the inner product with the weights, which is a linear algebra operation and can be done by quantum matrix multiplication (if the weights are dynamically encoded) or with a swap test (if they are amplitude encoded). The second step is the difficult part, namely the nonlinear update of the amplitudes. For example, to realise a step activation function, amplitudes with $|a_i|^2 > 0.5$ would have to be mapped to $\frac{1}{\sqrt{N}}$ where N is their number, and all other amplitudes have to vanish. An important question is therefore how to efficiently implement a nonlinear transformation $\mathbf{a} \to \varphi(\mathbf{a})$ on the amplitude vector $\mathbf{a}$. To assess this question, let us have a look at the options quantum theory offers:

A quantum evolution on a closed system, a quantum system with no interaction to its environment, only allows linear unitary transformations of the form $\mathbf{Ua}$, which in the density matrix formalism becomes $\mathbf{U}\rho\mathbf{U}^\dagger$ for $\rho = \mathbf{a}\mathbf{a}^\dagger$. As mentioned before, evidence shows that any nonlinear maps would enable us to solve NP-hard problems [88], allow for super-luminal communication [86] or negate the laws of thermodynamics [89] and can therefore be considered highly unlikely. In the theory of open quantum systems we look at a subsystem of a quantum state which allows us to perform a trace operation on the result. Open quantum systems can be described by completely positive trace preserving maps. Without digging too deep into the rich theory [228, 229] a completely positive trace preserving map on a density operator can be mathematically written as a sum of applications of a set of so called *Kraus operators* $\{K_i\}$ that can be expressed as matrices $\sum_i K_i \rho K_i^\dagger$, with $\sum_i K_i^\dagger K_i = 1$, which is again a linear transformation. One idea in this context is to use clever trace operations to approximate the series expansion of a tanh or sigmoid function up to a certain order, but this turns out to be nontrivial due to the structure of a trace operation.

A projective measurement P_k is clearly a nonlinear operation, but its outcome is stochastic and if averaging over several runs it will reflect the distribution of amplitudes (or a mixture of

potential measurement outcomes $\sum_k P_k \rho P_k$. The only exception is a conditional measurement for which the outcome k is ensured by repeating the algorithm until successful. This transforms ρ with certainty to the state $\frac{P_k \rho P_k}{\mathrm{tr} P_k \rho}$. As mentioned before, conditional measurements have to be analysed carefully regarding their success probability. The quantum gradient descent method has shown that consecutive application (as demanded by Design Principle 4) can increase the number of copies of a state exponentially with the number of layers. Furthermore, conditional measurements can often be replaced by classical rejection sampling, which leaves the fulfilment of Design Principle 6 open.

As the case of conditional measurements illustrates, there may be very well be tricks to implement the nonlinear map of a percepton in amplitude encoding. For example, effective nonlinear dynamics in quantum systems have been reported [230, 231, 232] and could in principle be exploited. However, this is at least a nontrivial enterprise. This insight extends to training routines, where nonconvex and nonlinear objective functions have to be minimised.[1]

10.3 Basis encoding and the quantum perceptron algorithm

The second option I want to explore here is to encode the inputs and outputs into the qubits' state itself as in the first two cases of Table 10.1. It was mentioned before that despite the extensive spatial resources, basis encoding has the advantage that there is always a quantum routine implementing $|\mathbf{x}\rangle|0\rangle \rightarrow |\mathbf{x}\rangle|f(\mathbf{x})\rangle$, even though more qubits as a computational overhead may be needed. This Section proposes a way of implementing the perceptron activation function in such a way, making use of the quantum phase estimation algorithm. More precisely, one constructs a Hamiltonian fulfilling the eigenvalue equation $\mathrm{e}^{iHt}|\mathbf{x}\rangle = \mathrm{e}^{i\mathbf{w}^T\mathbf{x}t}|\mathbf{x}\rangle$ and uses the quantum phase estimation routine to write the eigenvalue $\mathbf{w}^T\mathbf{x}$ into basis encoding. The first qubit of the eigenvalue register defines if the eigenvalue is larger or smaller than 0.5 and therefore simulates the result of a step function. Note that a coherent training method is still outstanding for future research.

Consider a state

$$|x_1 ... x_N\rangle|0...0\rangle,$$

where the second register is an index register for computation and contains τ qubits. The x_i are either represented by a qubit each in case of binary inputs, or approximated by τ qubits each in case of continuous features. For the following I consider binary features and outputs, in other words a perceptron model that maps from $\{0,1\}^n$ to $\{0,1\}$. Furthermore, consider weights in $[-1,1]$ and a step activation function

$$\varphi(z) = \begin{cases} 1, & \text{if } z \geq 0, \\ -1, & \text{else.} \end{cases} . \tag{10.2}$$

According to Table 10.1 there are two options to encode the weights: First, as a unitary evolution, and second, in basis encoding in a separate register. I will suggest a quantum routine for the former version first, and then suggest how to implement the latter.

[1] An interesting question is whether the quantum gradient descent method could be adapted to neural network training.

First, put the index register into a uniform superposition by using Hadamards on each qubit. leading to $\frac{1}{\sqrt{2^\tau}} \sum_{j=0}^{2^\tau-1} |x_1, ..., x_n\rangle|j\rangle$. Consider a unitary operator $U_{\mathbf{w}} = U_n(w_n)...U_2(w_2)U_1(w_1)U_0$ that is a sequence of single qubit unitary operators $U_i(w_i)$ acting on the ith qubit of the input register. The single-qubit unitaries have the form

$$U_k(w_k) = \begin{pmatrix} e^{-2\pi i w_k/2n} & 0 \\ 0 & e^{2\pi i w_k/2n} \end{pmatrix}.$$

U_0 adds a global phase of πi, so that the resulting phase of state $|x_1, ..., x_n\rangle|j\rangle$ is given by $\exp(2\pi i(\frac{1}{2n}\mathbf{w}^T\mathbf{x} + 0.5)) = \exp(2\pi i\phi)$.

For phase estimation, we need an oracle O that can perform

$$|\mathbf{x}\rangle|j\rangle \xrightarrow{\mathcal{O}} U_{\mathbf{w}}^j|\mathbf{x}\rangle|j\rangle,$$

for $j = 1...2^\tau$. For sufficiently small τ (which is given here as discussed later), this can always be constructed by applying U conditioned on the index register. For example, if the last qubit is in 1, apply U once, if the second last qubit is in 1, apply U twice, and if the kth last qubit is in 1 apply U for 2^{k-1} times.

Applying the oracle leads to a state

$$\frac{1}{\sqrt{2^\tau}} \sum_{j=0}^{2^\tau-1} |x_1, ..., x_n\rangle e^{2\pi i j\phi}|j\rangle.$$

The quantum phase estimation routine (introduced as Routine 3.4.2) applies an inverse quantum Fourier transform, and if $\phi \approx \frac{j}{2^\tau}$ and discard the input register, this results in a quantum state $|\phi\rangle$ that represents the phase ϕ in binary fraction encoding. For cases $\phi \neq \frac{j}{2^\tau}$, it can be shown that in order to obtain ϕ accurately up to k bits of precision with a success probability of $1 - \epsilon$, one has to choose $\tau = k + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ [68].

As a reminder, the qubits $q_1, ..., q_\tau$ of $|\phi\rangle$ represent the phase via the relationship $\phi \approx q_1\frac{1}{2^0} + \ldots + q_\tau\frac{1}{2^\tau}$. The first qubit therefore decides whether the phase is larger or smaller than 0.5, which translates into a decision of whether $\mathbf{w}^T\mathbf{x}$ is larger or smaller than 0. Since we are only interested in the value of the first qubit, a precision of only $\tau = 2$ lets us obtain a 85% probability of success, while only $\tau = 8$ qubits lead to a success probability of $99,9\%$.

The computational complexity of the quantum perceptron algorithm is comparable to resources for the n multiplications and single IF-operation needed to implement a classical perceptron, which are in $\mathcal{O}(n\tau)$. However, when we only look at the implementation of the nonlinear function, we require only $\frac{\tau(\tau+1)}{2} + 3\frac{\tau}{2}$ gates [68]. To get the first qubit accurate with probability $1 - \epsilon$, one needs $\tau = \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$.

With a slight variation, we can assume an additional register $|\mathbf{w}\rangle = |w_1^1, ..., w_1^\delta, \ldots, w_N^1, ..., w_N^\delta\rangle$ containing the weights in δ-bit binary fraction representation. To write the normalised net input

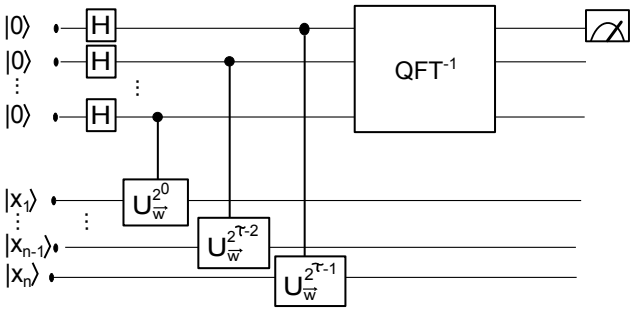


Figure 10.1: Quantum circuit for the quantum perceptron model.

$\mathbf{w}^T\mathbf{x}$ into the phase of quantum state $|\mathbf{x}\rangle|0...0\rangle|\mathbf{w}\rangle$ one has to replace the parameterised operator $U_\mathbf{w}$ with $\tilde{U} = U_0 \prod_{k=1}^{n} \prod_{m=1}^{\delta} U_{w_i^k, x_i}$ where U_0 again adds $1/2$ and we introduce the controlled two-qubit operators $U_{w_i^k, x_i}$ defined as

$$U_{w_i^k, x_i} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{\frac{-2\pi i}{2n2^k}} & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2n2^k}} \end{pmatrix}.$$

In words, the kth bit w_i^k of the binary representation of w_i controls a phase shift of 2^{-k}. Note that this implementation restricts the weights to $[0, 1)$, but a sign for each parameter can be stored in an additional qubit, and its inverse XOR with x_k can be used to control the sign of the phase shift.

Although this is a mere example of how to implement a perceptron in basis encoding, this algorithm opens up a couple of interesting avenues. For example, one can process training sets in superposition by starting with the state $\sum_{m=1}^{M} |\mathbf{x}^m\rangle$ instead of $|\mathbf{x}\rangle$, so that the entire procedure gets applied in parallel. The result would be a superposition of outputs. From measurements on the output qubit one could get the accuracy or success rate on an entire dataset, which could be useful for potential coherent training procedures.

The implementation of a step function can also be used as part of the ensemble method proposed in the next chapter. In summary, enhancing neural networks with quantum computing seems far more difficult than for many other methods.

Chapter 11

The Quantum ensemble learning algorithm

The results of this chapter have been written up for a manuscript with the title "Quantum parallelism for exponentially large ensemble classifiers" and will be submitted for publication soon. I was responsible for the idea, development, analysis and write-up of the content.

This last chapter of Part III investigates how *quantum parallelism* can be used to construct ensembles of quantum classifiers. *Ensemble methods* train a number of models in a specific training regime and use their combined decision to improve on each individual classifier. Quantum parallelism refers to the fact that a function $f(h)$ can be evaluated in superposition on a quantum device. More precisely, the operation $|h\rangle \otimes |0\rangle \rightarrow |h\rangle \otimes |f(h)\rangle$ on the qubit registers $|h\rangle, |0\rangle$ can be applied to a superposition of the first register, $\sum_h |h\rangle \otimes |0\rangle \rightarrow \sum_h |h\rangle \otimes |f(h)\rangle$. I will use this property in order to evaluate a 'superposition of quantum models' in parallel and extract a decision via a single qubit measurement. As a particular case, I will analyse a collective decision procedure where each model is weighed by its accuracy on the dataset. A quantum algorithm for the implementation of a quantum ensemble classifier is given, but the results from this analysis extend to applications with classical ensembles as well.

11.1 Classification with asymptotically large ensembles

Consider once more the supervised binary pattern classification problem: Given a dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(M)}, y^{(M)})\}$ with inputs $\mathbf{x}^{(m)} \in \mathbb{R}^N$ and outputs $y^{(m)} \in \{-1, 1\}$ with $m = 1, ..., M$, as well as a new input $\tilde{\mathbf{x}}$, predict the unknown output $\tilde{y}$. Consider furthermore a parametrised deterministic model (which I will focus on in this chapter) that can solve this task, formally given by

$$y = f(\mathbf{x}; \theta),$$

with input $\mathbf{x}$ and parameters θ. Machine learning methods choose a model by fitting the parameters to a dataset. Ensemble methods are based on the notion that allowing only one final model θ for prediction, whatever intricate the training procedure is, it will neglect the strengths of other candidates even if they have an overall worse performance. For example, one model might have learned how to deal with outliers very well, but at the expense of being slightly worse in

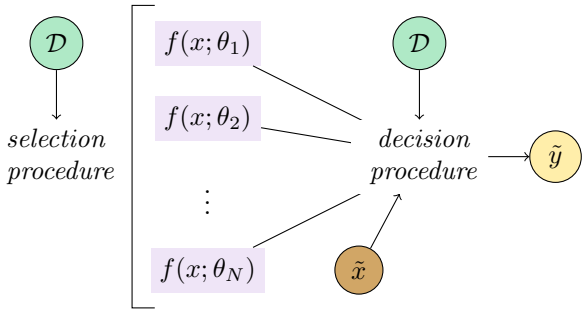


Figure 11.1: The principle of ensemble methods in machine learning is to select a set of classifiers and combine their predictions to obtain a better performance in generalising from the data than the best single model. Here, the N classifiers are considered to be deterministic parametrised model functions from a family $\{f(x;\theta)\}$, where the parameter θ solely defines the individual model. The dataset $\mathcal{D}$ is consulted in the selection procedure and sometimes also plays a role in the decision mechanism.

predicting the rest of the inputs. This 'expert knowledge' is lost if only one winner is selected. The idea is to allow for an ensemble or committee $\mathcal{E}$ of trained models (sometimes called 'experts' or simply 'classifiers') that take the decision for a new prediction together. Considering how familiar this principle is in our societies, it is surprising that this thought only gained widespread attention as late as the 1990s.

Many different proposals have been put forward of how to use more than one model for prediction.[1] The proposals can be categorised along two dimensions [233], first the selection procedure they apply to obtain the ensemble members, and second the decision procedure defined to compute the final output (see Figure 11.1). The easiest strategy would possibly be to train several models and decide according to their majority rule. More intricate variations are popular in practice and have interesting theoretical foundations. *Bagging* [234] trains classifiers on different subsamples of the training set, thereby reducing the variance of the prediction. *AdaBoost* [235, 236] trains subsequent models on the part of the training set that was misclassified previously and applies a given weighing scheme, which reduces the bias of the prediction. *Mixtures of experts* train a number of classifiers using a specific error function and in a second step train a 'gating network' that defines the weighing scheme [237]. For all these methods, the ensemble classifier can be written as

$$\tilde{y} = \operatorname{sgn}\left(\sum_{\theta\in\mathcal{E}} w_\theta f(\tilde{x};\theta)\right). \tag{11.1}$$

The coefficients w_θ weigh the decision $f(\tilde{x};\theta) \in \{-1, 1\}$ of the model specified by $\theta \in \mathcal{E}$, while the sign function assigns class 1 to the new input if the weighed sum is positive and -1 otherwise. It is important for the following to rewrite this as a sum over all E *possible* parameters. Here I will use a finite representation of numbers and limit the parameters to a certain interval to get the discrete sum

$$\tilde{y} = \operatorname{sgn}\left(\sum_{\theta=0}^{E-1} w_\theta f(\tilde{x};\theta)\right). \tag{11.2}$$

[1] Here I will focus on the idea of using a parametrised model family with fixed hyperparameters. In more general, ensembles can be composed of models with different hyperparameters (for example neural networks of a different architecture), or even different model types all together (for example neural networks, random forests and linear regression).

In order to obtain the ensemble classifier of Equation 11.1, the weights w_θ which correspond to models that are not part of the ensemble $\mathcal{E}$ are set to zero. Given a model family f, an interval for the parameters as well as a precision to which they are represented, an ensemble is specified by the set of weights $\{w_1 ... w_E\}$.

Writing a sum over all possible models provides a framework to think about asymptotically large ensembles which can for example be realised by quantum parallelism. Interesting enough, this formulation is also very close to another paradigm of classification, the Bayesian learning approach presented in Section 2.2. This method uses the knowledge represented by the data to estimate which hypothesis θ is the 'true' one. If we replace $f(\tilde{\mathbf{x}}; \theta)$ in Equation (11.2) with a probabilistic distribution $p(\tilde{y}|\tilde{\mathbf{x}}, \theta)$ and interpret w_θ as the likelihood $p(\theta|\mathcal{D})$ of θ being the true model given the observed data, we arrive the a Bayesian classification rule.

$$p(\tilde{y}|\tilde{\mathbf{x}}, \mathcal{D}) = \int d\theta \;\; p(\tilde{y}|\tilde{\mathbf{x}}, \theta) p(\theta|\mathcal{D}). \tag{11.3}$$

The probability of the new input $\tilde{\mathbf{x}}$ being in class $\tilde{y} = 1$ given the data can be expressed as the integral over the probability of a prediction of 1 given the model, times the probability of this particular model given the data. If we also consider different model families specified by hyperparameters, this method turns into *Bayesian Model Averaging* which is sometimes considered as an ensemble method (although not necessarily leading to optimal generalisation results [238]).

Beyond the Bayesian framework, increasing the size of the ensemble to include all possible parameters has been studied in different contexts. In some cases adding *accurate* classifiers with a classification error (the number of misclassified test examples divided by their total) of less than 0.5 increases the performance of the ensemble decision. This means that each member is better than random guessing and has learned the pattern of the training set to at least a small extend. The most well-known case has been found by Schapire [235] leading to the aforementioned AdaBoost algorithm where a collection of such weak classifiers can be turned into a strong classifier that has a high accuracy on the test set. The advantage here is that weak classifiers are comparably easy to train and combine. But people thought about the power of weak learners long before AdaBoost. The *Cordocet Jury Theorem* from 1758 states that considering a committee of judges where each judge has a probability p with $p > 0.5$ to reach a correct decision, the probability of a correct collective decision by majority vote will converge to 1 as the number of judges approaches infinity. This idea has been applied to ensembles of neural networks by Hansen and Salamon [239]. If all ensemble members have a likelihood of p to classify a new instance correctly (which can be estimated by their accuracy a on the test set), and their errors are uncorrelated, the probability that the majority rule classifies the new instance incorrectly is given by

$$\sum_{k>E/2}^{E} \binom{E}{k} p^{E-k}(1-p)^k,$$

where E is again the ensemble size. The convergence behaviour is plotted in Figure 11.2 (left) for different values of p. The assumption of uncorrelated errors is idealistic, since some data points will be more difficult to classify than others and therefore tend to be misclassified by a large proportion

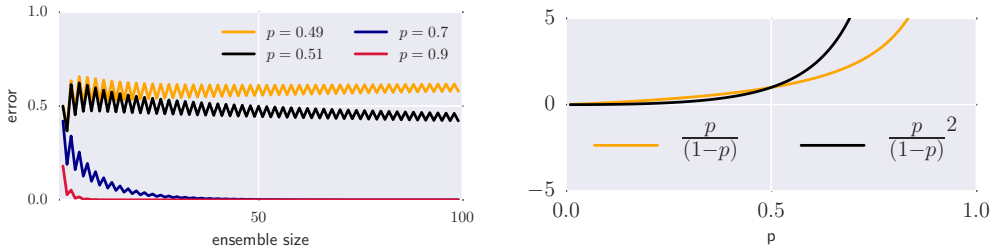


Figure 11.2: Left: Prediction error when increasing the size of an ensemble of classifiers each of which has an accuracy p. Asymptotically, the error converges to zero if $p > 0.5$. Right: For $p > 0.5$, the odds ratio $p/(1-p)$ grows slower than its square. Together with the results from Lam et al described in the text, this is an indication that adding accurate classifiers to an ensemble increases its predictive power.

of the ensemble members. Hansen and Salamon argue that for the highly overparametrised neural network models they consider as base classifiers, training will get stuck in different local minima, so that the ensemble members will be diverse and their errors sufficiently uncorrelated.

A more realistic setting would also assume that each model has a different prediction probability p measured by the accuracy a, which has been investigated by Lam and Suen [240]. The change in prediction power with the growth of the ensemble obviously depends on the predictive power of the new ensemble member, but its sign can be determined. Roughly stated, adding two classifiers with accuracies a_1, a_2 to an ensemble of size $2n$ will increase the prediction power if the value of $\frac{a_1 a_2}{(1-a_1)(1-a_2)}$ is not less than the ratio $\frac{a_i}{(1-a_i)}$ of any ensemble member, $i = 1, ..., E$. When plotting the ratio and its square in Figure 11.2 (right), it becomes apparent that for all $a_i > 0.5$ chances are high to increase the predictive power of the ensemble. All these results suggest that constructing large ensembles of accurate classifiers can lead to a strong combined classifier.

Before proceeding to quantum models, another result is important to mention. If we consider all possible parameters θ in the sum of Equation (11.2) and assuming that the model defined by θ has an accuracy a_θ on the training set, the optimal weighing scheme is given by

$$w(\theta) = \log \frac{a_\theta}{1 - a_\theta}. \tag{11.4}$$

It is interesting to note that this weighing scheme corresponds to the weights chosen in AdaBoost for each trained model, where they are derived from what seems to be a different theoretical objective.

11.2 Quantum ensembles of quantum classifiers

The idea of this section is to cast the notion of ensembles into a quantum algorithmic framework, based on the idea that quantum parallelism can be used to evaluate ensemble members of an exponentially large ensemble in one step. Consider a quantum routine $\mathcal{A}$ which 'computes' a model function $f(\mathbf{x}; \theta)$,

$$\mathcal{A}\,|\mathbf{x}\rangle|\theta\rangle|0\rangle \rightarrow |\mathbf{x}\rangle|\theta\rangle|f(\mathbf{x};\theta)\rangle,$$

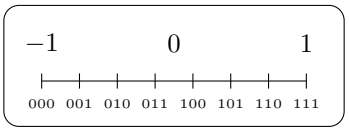


Figure 11.3: Example for the representation of parameters in a uniform quantum superposition of three qubits. Each computational basis state $|000\rangle, |001\rangle, ..., |111\rangle$ corresponds to a parameter in the interval $[-1, 1]$.

which I will call a *quantum classifier* in the following. The last qubit $|f(\mathbf{x};\theta)\rangle$ thereby encodes class $f(\mathbf{x};\theta) = -1$ in state $|0\rangle$ and class 1 in state $|1\rangle$. The quantum registers can encode the classical vectors $\mathbf{x}, \theta$ in either amplitude or basis encoding. Note that in principle, every function $f(\mathbf{x};\theta)$ a classical computer can compute efficiently could be translated into a basis encoded quantum circuit $\mathcal{A}$, which means that every classifier leads to an efficient quantum classifier (possibly with large polynomial overhead). An example would be the perceptron algorithm presented in a previous chapter. The important point is that $\mathcal{A}$ could be implemented in parallel to a superposition of parameter states.

$$\mathcal{A}\,|\mathbf{x}\rangle\,\frac{1}{\sqrt{E}}\sum_{\theta}|\theta\rangle\,|0\rangle \rightarrow |\mathbf{x}\rangle\,\frac{1}{\sqrt{E}}\sum_{\theta}|\theta\rangle\,|f(\mathbf{x};\theta)\rangle.$$

For example, $\frac{1}{\sqrt{E}}\sum_{\theta}|\theta\rangle$ could be a uniform superposition,

$$\frac{1}{\sqrt{E}}\sum_{i=0}^{2^{\tau}-1}|i\rangle,$$

which prepares a superposition of all τ-bit length binary representations of the parameters. The superposition equally divides the space of possible parameters into 2^{τ} candidates (see Figure 11.3).

As explained before, an ensemble method can be understood as a weighing rule for each model in the sum of Equation (11.2). We will therefore require a second quantum routine, $\mathcal{W}$, which turns the uniform superposition into a non-uniform one,

$$\mathcal{W}\;\;|\mathbf{x}\rangle\,\frac{1}{\sqrt{E}}\sum_{\theta}|\theta\rangle|0\rangle \rightarrow |\mathbf{x}\rangle\,\frac{1}{\sqrt{E\chi}}\sum_{\theta}\sqrt{w_{\theta}}|\theta\rangle|0\rangle,$$

weighing each model θ by a classical probability w_{θ}. I will call this routine a *quantum ensemble* since the weights define which model contributes to what extend to the combined decision. The normalisation factor χ ensures that $\sum_{\theta}\frac{w_{\theta}}{E\chi} = 1$. Note that this routine can be understood as a state preparation protocol of a qsample.

Together, the quantum routines $\mathcal{A}$ and $\mathcal{W}$ define the quantum ensemble of quantum classifiers, by first weighing the superposition of models (in other words the basis states in the parameter register) and then computing the model predictions for the new input $\tilde{\mathbf{x}}$ in parallel. The final quantum state is given by

$$|\tilde{\mathbf{x}}\rangle\frac{1}{\sqrt{E\chi}}\sum_{\theta}\sqrt{w_{\theta}}|\theta\rangle|f(\tilde{\mathbf{x}};\theta)\rangle.$$

The measurement statistics of the last qubit contains the ensemble prediction: The chance of

measuring the qubit in state 0 is the probability of the ensemble deciding for class -1 and is given by

$$p(\tilde{y}=-1)=\sum_{\theta\in\mathcal{E}^+}\frac{w_\theta}{E\chi},$$

while the chance of measuring the qubit in state 1 reveals $p(\tilde{y}=1)$ and is

$$p(\tilde{y}=1)=\sum_{\theta\in\mathcal{E}^-}\frac{w_\theta}{E\chi},$$

where $\mathcal{E}^\pm$ is the subset of $\mathcal{E}$ containing only models with $f(\tilde{\mathbf{x}};\theta)=\pm 1$. After describing the general template, I will look at how to implement a specific weighing scheme with a quantum routine $\mathcal{W}$ for general models $\mathcal{A}$ and analyse the resulting classifier in detail.

11.3 Choosing the weights proportional to the accuracy

The quantum ensemble classifier that will be analysed in more detail uses weights that are proportional to the accuracy a_θ of each model on the data set, which is the proportion of correct classifications

$$a_\theta=\frac{1}{M}\sum_{m=1}^{M}\frac{1}{2}|f(x^m;\theta_i)+y^m|.$$

Note that similar to Bayesian learning, we do not require a separate test set for the training of individual models any more. The goal of the quantum algorithm is to prepare a quantum state where each model $|\theta\rangle$ has an amplitude proportional to its accuracy, $w_\theta\propto a_\theta$.

The weighing routine $\mathcal{W}$ can be constructed as follows. Required for computation is a system of $(\delta+1)+\tau+1+1$ qubits divided into four registers: the data register, the parameter register, the output register and the accuracy register,

$$\underbrace{|0...0;0\rangle}_{\delta+1\ \text{qubits}}\otimes\underbrace{|0...0\rangle}_{\tau\ \text{qubits}}\otimes|0\rangle\otimes|0\rangle. \tag{11.5}$$

Assume a quantum classification routine $\mathcal{A}$ is given. As a first step, τ Hadamards bring the parameter register into a uniform superposition, and a Hadamard is applied to the accuracy qubit:

$$\frac{1}{\sqrt{E}}\sum_{i=0}^{2^\tau-1}|0...;0\rangle|i\rangle|0\rangle\ \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle).$$

Each $|i\rangle$ thereby encodes a set of parameters θ. We now 'load' the training inputs successively into the data register, compute the outputs in superposition by applying the core routine $\mathcal{A}$ and rotate the accuracy qubit by a small amount towards $|0\rangle$ or $|1\rangle$ depending on whether target output and actual output have the same value (i.e. by a XOR gate on the respective qubits together with a conditional rotation of the last qubit). The core routine and loading step are then uncomputed by applying the inverse operations. Note that one could alternatively prepare a training superposition $\frac{1}{\sqrt{M}}\sum_m|\mathbf{x}^m\rangle$ and trace the training register out in the end. The costs remain linear in the number of training vectors times the bit-depth for each training vector for both strategies.

After all training points have been processed, the accuracy qubit is entangled with the parameter register and in state $|q\rangle = \sqrt{a_\theta}|0\rangle + \sqrt{1-a_\theta}|1\rangle$. A postselective measurement on the accuracy qubit only accepts when it is in in state $|0\rangle$ and repeats the routine otherwise. This serves to select the $|0\rangle$-branch of the superposition and leaves us with the state

$$\frac{1}{\sqrt{E\chi}} \sum_\theta \sqrt{a_\theta}|0...;0\rangle|\theta\rangle|0\rangle,$$

where the normalisation factor χ is equal to the acceptance probability $p_{\text{acc}} = \frac{1}{E}\sum_\theta a_\theta$. As discussed before, the probability of acceptance influences the runtime of the algorithm, since a measurement of the ancilla in 1 means we have to abandon the result and start the routine from scratch. In this case, the a_θ can be expected to be distributed around 0.5 since we expect most models to have an accuracy close to random guessing, and postselection has a high chance to succeed. This might hint towards rejection sampling as a promising tool to translate the accuracy-weighted quantum ensemble into a classical method.

Now load the new input into the first δ qubits of the data register, apply the routine $\mathcal{A}$ once more and uncompute (and disregard) the data register to obtain

$$|\psi\rangle = \frac{1}{\sqrt{E\chi}} \sum_\theta \sqrt{a_\theta}|\theta\rangle|f(\tilde{\mathbf{x}},\theta)\rangle.$$

The measurement statistics of the last qubit now contain the desired value. More precisely, the expectation value of $\mathbb{I}\otimes\sigma_z$ is given by

$$\langle\psi|\mathbb{I}\otimes\sigma_z|\psi\rangle = \frac{1}{E\chi}\sum_\theta a_\theta(f(\tilde{x};\theta)+1)/2,$$

and corresponds to the classifier in Equation 11.2 (note that $f(\tilde{\mathbf{x}};\theta) \in \{-1,1\}$ has to be mapped to the values $\{0,1\}$ of a qubit). Repeated measurements reveal this expectation value to the desired precision.

11.4 Why accuracies may be good weights

I will now turn to the question why the accuracies might be a good weighing scheme. Recall that there is a lot of evidence that ensembles of weak but accurate classifiers (meaning that $a_\theta > 0$ for all θ) can lead to a strong classifier. The ensemble constructed in the last section however contains all sorts of models which did not undergo a selection procedure, and it may therefore contain a large –or even exponential– share of models with low accuracy or random guessing. It turns out that for a large class of model families, the ensemble *effectively* only contains accurate models (i.e. with accuracy > 0.5). This is to my knowledge a new result also interesting for classical ensemble methods.

Assume the core machine learning model has the property to be point symmetric in the parameters θ,

$$f(x;-\theta) = -f(x;\theta).$$

This is true for linear models and neural networks with an odd number of layers such as a simple

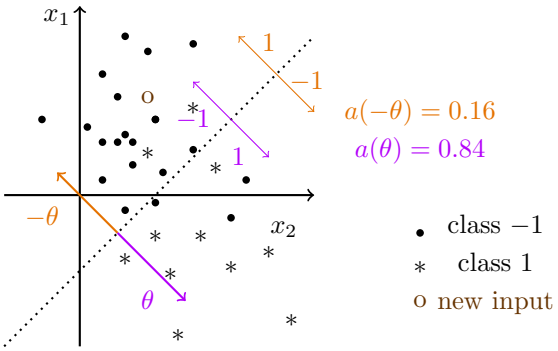


Figure 11.4: Illustration of the decision boundary of a perceptron or neural network model. The parameter vector defines a hyperplane dividing input space into regions of class −1 and 1. A change of sign of the parameters swaps the decision regions.

perceptron or an architecture with 2 hidden layers. Let us furthermore assume that the parameter space Θ is symmetric around zero, meaning that for each $\theta \in \Theta$ there is also $-\theta \in \Theta$. These pairs are denoted by $\theta^+ \in \Theta^+, \theta^- \in \Theta^-$. With this notation one can write

$$f(x;\theta^+) = -f(x;\theta^-).$$

From there it follows that $\int_{\theta\in\Theta} f(x;\theta) = 0$ and $a(\theta^+) = 1 - a(\theta^-)$.

To get an intuition, consider a linear model imposing a linear decision boundary in the input space. The parameters indicate the vector orthogonal to the decision boundary (in addition to a bias term that defines where the boundary intersects with the y-axis). A sign change of all parameters flips the vector around; the linear decision boundary remains at exactly the same position, meanwhile all decisions are turned around (see Figure 11.4).

For point symmetric models the expectation can be expressed as a sum over one half of the parameter space:

$$\frac{1}{E\chi}\sum_{\theta} a_\theta f(\tilde{x},\theta) = \frac{1}{E\chi}\sum_{\theta^+}\left[a(\theta^+) - \frac{1}{2}\right] f(\tilde{x},\theta^+). \tag{11.6}$$

The effect of this 'phantom transformation' is to shift the weights from the interval $[0,1]$ to $[-0.5, 0.5]$, with profound consequences. The transformation is plotted in Figure 11.5. One can see that accurate models get a positive weight, while non-accurate models get a negative weight and random guessers vanish from the sum. The negative weight flips the decision $f(\mathbf{x}:\theta)$ of the 'bad' models. The more structure a model recognises in the data, the higher the weight. This is a linearisation of the rule introduced in Equation (11.4) as the optimal weight distribution for large ensembles (plotted in black for comparison).[2]

With this in mind we can rewrite the expectation value as

$$\mathbb{E}[f(\tilde{x};\theta)] = \frac{1}{E}\sum_{\theta|a(\theta)>0.5} \tilde{a}(\theta) f(\tilde{x},\theta), \tag{11.7}$$

[2]By constructing a quantum routine that instead of turning an accuracy qubit by a value proportional to the accuracy, but proportional to $\log\frac{a}{1-a}$, we could even recover the optimal weight distribution for the quantum algorithm.

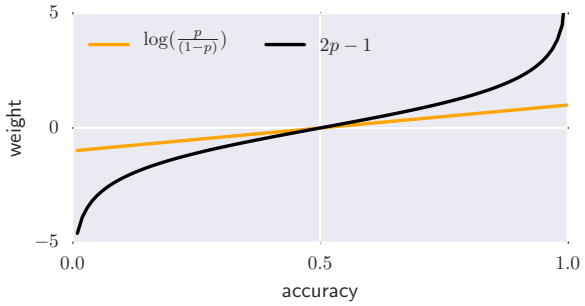


Figure 11.5: Comparison of the effective weight transformation and the optimal weight rule. Both flip the prediction of a classifier that has an accuracy of less than 0.5.

with the new weights

$$\tilde{a}(\theta) = a(\theta) - \frac{1}{2}. \tag{11.8}$$

This 'transformation' is valid for parameters θ^+ as well as θ^-.

If one cannot assume a point-symmetric parameter space but still wants to construct an ensemble of all accurate models (i.e., $a_\theta > 0.5$), an alternative to the above sketched routine $\mathcal{W}$ could be to construct an oracle that marks such models and then use amplitude amplification to create the desired distribution. A way to mark the models is to load every training vector into a predefined register and compute $|f(\mathbf{x}^m;\theta)\rangle$ as before, but perform a binary addition on a separate register that "counts" the number of correctly classified training inputs. The register would be a binary encoding of the count, and hence require $\lceil \log M \rceil$ qubits (as well as garbage qubits for the addition operation). If $\log M = \mu$ for an integer μ, then the first qubit would be one if the number of correct classifications is larger than $M/2$ and zero otherwise. In other words, this qubit would flag the accurate models and can be used as an oracle for amplitude amplification. The optimal number of Grover iterations depends on the number of flagged models. In the best case, around $\frac{1}{4}$th of all models are accurate so that the optimal number of iterations is approximately $\sqrt{E/\frac{1}{4}E} = 2$. Of course, this number has to be estimated before performing the Grover iterations.[3]

11.5 Analytical investigation of the accuracy-weighted ensemble

In order to explore the accuracy-weighted ensemble classifier further, I want to conduct some analytical investigations. It is convenient to assume that we know the probability distribution $p(x, y)$ from which the data is picked (that is either the 'true' probability distribution with which data is generated, or the approximate distribution inferred by some data mining technique). Furthermore, we consider the continuous limit $\sum \to \int$. Each parameter θ defines decision regions in the input space, $\mathcal{R}_{-1}^{\theta_i}$ for class -1 and $\mathcal{R}_1^\theta$ for class 1 (i.e. regions of inputs that are mapped to the respective

[3] If one can analytically prove that $\frac{1}{2}E$ of all possible models will be accurate, one can artificially extend the superposition to twice the size, prevent half of the subspace from being flagged and thereby achieve the optimal amplitude amplification scheme.

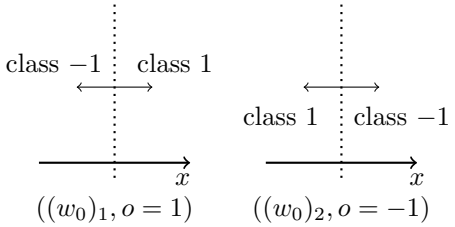


Figure 11.6: Very simple model of a 1-dimensional classifier analysed in the text: The parameter w_0 denotes the position of the decision boundary while parameter o determines its orientation.

classes). The accuracy can then be expressed as

$$a(\theta) = \frac{1}{2} \int_{\mathcal{R}_{-1}^{\theta}} p(x, y = -1)\, dx + \frac{1}{2} \int_{\mathcal{R}_{1}^{\theta}} p(x, y = 1)\, dx. \tag{11.9}$$

In words, this expression measures how much of the density of a class falls into the decision region proposed by the model for that class. Good parameters will propose decision regions that contain the high-density areas of a class distrubution. The factor of $1/2$ is necessary to ensure that the accuracy is always in the interval $[0, 1]$ since the two probability densities are each normalised to 1.

The probability distributions we consider here will be of the form $p(x, y = \pm 1) = g(x; \mu_\pm, \sigma_\pm) = g_\pm(x)$. They are normalised, $\int_{-\infty}^{\infty} g_\pm(x)dx = 1$, and vanish asymptotically, $g_\pm(x) \to 0$ for $x \to -\infty$ and $x \to +\infty$. The hyperparameters μ_-, σ_- and μ_+, σ_+ define the mean or 'position' and the variance or width of the distribution for the two classes -1 and 1. Prominent examples are Gaussians or step functions. Let $G(x; \mu_\pm, \sigma_\pm) = G_\pm(x)$ be the integral of $g(x; \mu_\pm, \sigma_\pm)$, which will fulfils $G_\pm(x) \to 0$ for $x \to -\infty$ and $G_\pm(x) \to 1$ for $x \to +\infty$. Two expressions following from this property which will become important later are

$$\int_{-\infty}^{a} g_\pm(x)dx = G_\pm(a) - G_\pm(-\infty) = G_\pm(a),$$

and

$$\int_{a}^{\infty} g_\pm(x)dx = G_\pm(\infty) - G_\pm(a) = 1 - G_\pm(a).$$

I consider a minimal toy example for a classifier, namely a perceptron model on a one-dimensional input space, $f(x; w, w_0) = \mathrm{sgn}(wx + w_0)$ with $x, w, w_0 \in \mathbb{R}$. While one parameter would be sufficient to mark the position of the point-like 'decision boundary', a second one is required to define its orientation. One can simplify the model even further by letting the bias w_0 define the position of the decision boundary and introducing a binary 'orientation' parameter $o \in \{-1, 1\}$ (as illustrated in Figure 11.6),

$$f(x; o, w_0) = \mathrm{sgn}(o(x - w_0)). \tag{11.10}$$

For this simplified perceptron model the decision regions are given by

$$\mathcal{R}_{-1} = [-\infty, w_0], \mathcal{R}_1 = [w_0, \infty],$$

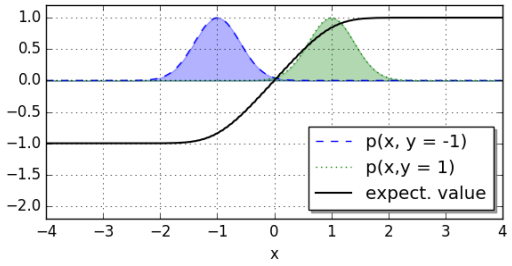


Figure 11.7: The black line plots the expectation value for the blue and green class densities for different new inputs $\tilde{x}$. A positive expectation value yields the prediction 1 while a negative expectation value predicts class -1. At point zero lies the decision boundary. The plot shows that the model predicts the decision boundary where we would expect it, namely between the two distributions

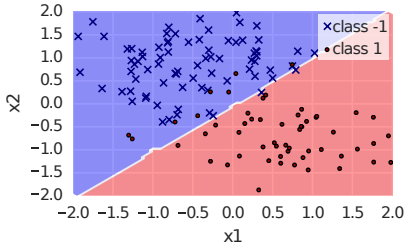


Figure 11.8: Perceptron classifier with 2-dimensional inputs and a bias. The ensemble consist of 8000 models, each of the three parameters taking 20 equally distributed values from $[-1, 1]$. The resolution to compute the decision regions was $\Delta x_1 = \Delta x_2 = 0.05$. The dataset was generated with python scikit-learn's blob function and both classes have the same variance parameter. One can see that in two dimensions, the decision boundary still lies in between the two means of $\mu_{-1} = [-1, 1]$ and $\mu_1 = [1, -1]$.

for the orientation $o = 1$ and

$$\mathcal{R}_{-1} = [w_0, \infty], \mathcal{R}_1 = [-\infty, w_0],$$

for $o = -1$. Our goal is to compute the expectation value

$$\mathbb{E}[f(\tilde{x}; w_0, o)] \propto \int d\theta \; a(\theta) f(\tilde{x}; o, w_0),$$

of which the sign function evaluates the desired prediction $\tilde{y}$.

Inserting the definitions from above, as well as some calculus using the properties of $p(x)$ brings this expression to

$$\int_{-\infty}^{\infty} dw_0 \; (G_-(w_0) - G_+(w_0)) \; \mathrm{sgn}(\tilde{x} - w_0),$$

and evaluating the sign function for the two cases $\tilde{x} > w_0, \tilde{x} < w_0$ leads to

$$\int_{-\infty}^{\tilde{x}} dw_0 \Big\{ G_+(w_0) - G_-(w_0) \Big\} - \int_{\tilde{x}}^{\infty} dw_0 \Big\{ G_+(w_0) - G_-(w_0) \Big\}. \tag{11.11}$$

To analyse this expression further, consider the two class densities to be Gaussian probability distributions

$$p(x, y = \pm 1) = \frac{1}{\sigma_\pm \sqrt{2\pi}} \exp^{-\frac{(x-\mu_\pm)^2}{2\sigma_\pm^2}}.$$

The indefinite integral over the Gaussian distributions is given by

$$G_\pm(x) = \frac{1}{2}[1 + \mathrm{erf}\frac{x - \mu_\pm}{\sqrt{2}\sigma_\pm}].$$

Inserting this into Eq (11.11) we get

$$\int_{-\infty}^{\tilde{x}} dw_0 \Big\{ \mathrm{erf}\frac{w_0 - \mu_+}{\sqrt{2}\sigma_+} - \mathrm{erf}\frac{w_0 - \mu_-}{\sqrt{2}\sigma_-} \Big\} - \int_{\tilde{x}}^{\infty} dw_0 \Big\{ \mathrm{erf}\frac{w_0 - \mu_+}{\sqrt{2}\sigma_+} - \mathrm{erf}\frac{w_0 - \mu_-}{\sqrt{2}\sigma_-} \Big\}. \quad (11.12)$$

Figure 11.7 plots the expectation value for different inputs $\tilde{x}$ for the case of two Gaussians with $\sigma_- = \sigma_+ = 0.5$, and $\mu_- = -1, \mu_+ = 1$. The decision boundary is at the point where the expectation value changes from a negative to a positive number, $\mathbb{E}[f(\hat{x}; w_0, o)] = 0$. One can see that for this simple case, the decision boundary will be in between the two means, which we would naturally expect. This is an important finding, since it implies that the accuracy-weighted ensemble classifier works - arguebaly only for a very simple model and dataset. A quick calculation shows that we can expect to find the decision boundaries midway between the two means for all cases with $\sigma_- = \sigma_+$. In this case the integrals in Equation (11.12) evaluate to

$$\int_{-\infty}^{\tilde{x}} dw_0 \ \mathrm{erf}\frac{w_0 - \mu_\pm}{\sqrt{2}\sigma_\pm} = \gamma_\pm(\tilde{x}) - \lim_{R \to -\infty} \gamma_\pm(R),$$

and

$$\int_{\tilde{x}}^{\infty} dw_0 \ \mathrm{erf}\frac{w_0 - \mu_\pm}{\sqrt{2}\sigma_\pm} = \lim_{R \to \infty} \gamma_\pm(R) - \gamma_\pm(\tilde{x}),$$

with the integral over the error function

$$\gamma_\pm(x) = (x - \mu_\pm)\ \mathrm{erf}\left(\frac{x - \mu_\pm}{\sqrt{2}\sigma_\pm}\right) + \sqrt{\frac{2}{\pi}}\sigma_\pm \mathrm{e}^{-(\frac{x-\mu_\pm}{\sqrt{2}\sigma_\pm})^2}.$$

Assuming that the mean and variance of the two class distributions are of reasonable (i.e. finite) value, the error function evaluates to 0 or 1 before the limit process becomes important, and one can therefore write

$$\lim_{R \to -\infty} \gamma_\pm(R) = \lim_{R \to -\infty} \sqrt{\frac{2}{\pi}}\sigma_\pm \mathrm{e}^{-(\frac{R-\mu_\pm}{\sqrt{2}\sigma_\pm})^2}, \quad (11.13)$$

$$\lim_{R \to \infty} \gamma_\pm(R) = \lim_{R \to \infty} R + \lim_{R \to \infty} \sqrt{\frac{2}{\pi}}\sigma_\pm \mathrm{e}^{-(\frac{R-\mu_\pm}{\sqrt{2}\sigma_\pm})^2} - \mu. \quad (11.14)$$

The expectation value for the case of equal variances therefore becomes

$$\mathbb{E}[f(\tilde{x}; w_0, o)] = 2\gamma_-(\tilde{x}) - 2\gamma_+(\tilde{x}). \quad (11.15)$$

Setting $\tilde{x} = \hat{\mu} = \mu_- + 0.5(\mu_+ + \mu_-)$ turns the expectation value to zero; the point $\hat{\mu}$ between

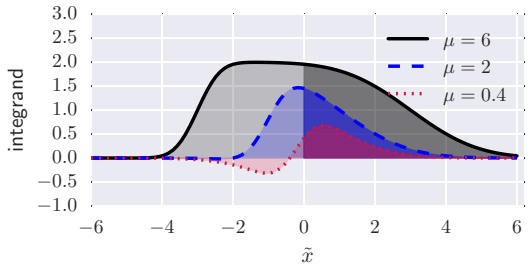


Figure 11.9: Plot of the integrand in Equation (11.12), $I = \operatorname{erf}\frac{x-\mu_+}{\sqrt{2}\sigma_+} - \operatorname{erf}\frac{x-mu_-}{\sqrt{2}\sigma_+}$, for $\sigma_- = 0.5, \sigma_+ = 1.5$ and different distances $\mu = \mu_- - \mu_+$ of the means whose average value is chosen to be zero. A higher variance accounts for a 'flatter' flank of the function. When the means of the data distributions have a sufficient distance, the area under the curve right of 0 is equal to the area under the curve left of 0, and the resulting decision boundary lies exactly between the means. If the two distributions are too close together, this symmetry vanishes for $\sigma_- \neq \sigma_+$. In the case $\sigma_- = \sigma_+$ the symmetry is always given.

the two variances is shown to be the decision boundary. Simulations confirm that this is also true for other distributions, such as a square, exponential or Lorentz distribution, as well as for two-dimensional data (see Figure 11.8).

To understand what happens in the case of different variances, the integrand of Equation (11.12) is plotted in Figure 11.9 for different distances $\mu = \mu_+ - \mu_-$. The value of $\sigma_\pm$ changes the steepness of the flanks of the distribution. As long as the means of the distributions are far enough apart from one another, this does not effect the decision boundary lying in their middle (since the steepness of the flank does not affect the area under the curve for positive or negative x-values). However, for certain small values of μ the distribution does not carry the same area on the right and left side of $x = 0$.

The simplicity of the core model allows us to have a deeper look into the structure of the expectation value. Figure 11.10 shows the different component functions of the ensemble classifier over the weight space, namely the accuracy, the core model function as well as their product, for the examples $\sigma_\pm = 1$, $\tilde{x} = 0.7$, the two cases $o = -1, 1$ and different means. One can see that the final function over which we integrate in order to obtain the expectation value (marked in grey) is point symmetric if $\tilde{x}$ lies in the middle of the class distributions' means. At this point, the expectation value vanishes which defines the location of the decision boundary. One can see that for two class distributions close to each other, the accuracy function is only symmetric if their variance is the same. One can conclude that the function of the accuracy over the parameter plays an important role for the location of the decision boundary. As a summary, the analytical considerations give evidence that the weighted average ensemble classifier works well with linear models and for simple data distributions.

Further investigations would have to show whether the accuracy-weighted ensemble classifier is able to realise nonlinear decision boundaries if the base classifiers are more flexible. This is a huge computational effort, since for example the next more complex neural network model requires two hidden layers to be point symmetric, and with one bias neuron and for two-dimensional inputs

the architecture has already seven or more weights. If each weight is encoded in three qubits only (including one qubit for the sign), we get an ensemble of 2^{21} members whose collective decision needs to be computed for an entire input space in order to determine the decision boundaries. However, low resolution simulations with one-dimensional inputs confirm that nonlinear decision boundaries can be learnt. Sampling methods could help to obtain approximations to this result.

Another important question concerns overfitting. In AdaBoost, regularisation is equivalent to early stopping [241]. Bayesian learning does not encounter problems of overfitting. Simple models like the perceptron considered here may have an 'inbuilt' regularisation mechanism. However, more flexible models considered for the accuracy-weighted ensemble may very well suffer from overfitting.

A third avenue of further investigation is to look for speedups in the quantum ensemble models, for example by constructing weighting schemes that are difficult to implement with classical sampling methods, or by applying pure quantum classifiers.

In summary, this chapter introduced the quantum ensemble framework as a template for collective decisions of quantum models. Going beyond the quest for speedups from earlier chapters, this work demonstrates that thinking about quantum algorithms can lead to new insights into models that can also be interesting from the perspective of classical computation.

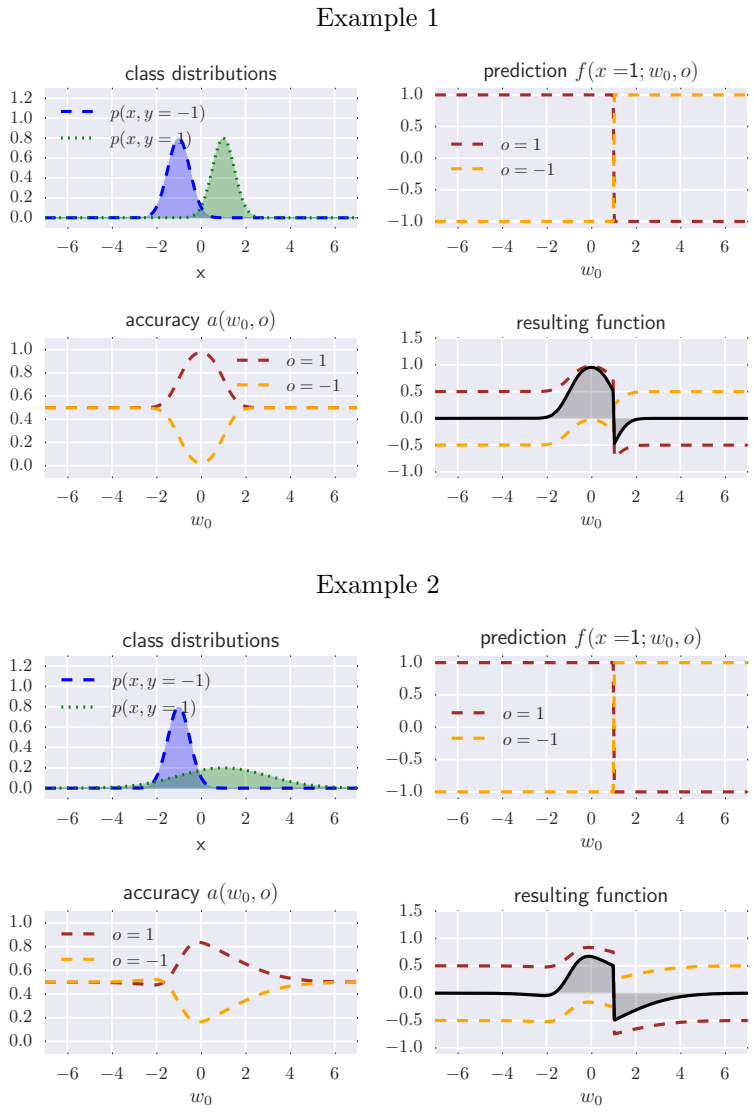


Figure 11.10: Detailed analysis of the classification of $\tilde{x} = 0.7$ for two examples of data distributions. The upper Example 1 shows $p(x, y = -1) = \mathcal{N}(-1, 0.5)$ and $p(x, y = 1) = \mathcal{N}(1, 0.5)$ while the lower Example 2 shows $p(x, y = -1) = \mathcal{N}(-1, 0.5)$ and $p(x, y = 1) = \mathcal{N}(1, 2)$ (plotted each in the top left figure of the four). The top right figure in each block shows the classification of a given new input $\tilde{x}$ for varying parameters b, plotted for $o = 1$ and $o = -1$ respectively. The bottom left shows the accuracy or classification performance $a(w_0, o = 1)$ and $a(w_0, o = -1)$ on the data distribution. The bottom right plots the product of the previous two for $o = 1$ and $o = -1$, as well as the resulting function under the integral. The prediction outcome is the integral over the black curve, or the total of the gray shaded areas. One can see that for models with different variances, the accuracies loose their symmetry and the decision boundary will therefore not lie in the middle between the two means.

Chapter 12

Conclusion

This thesis investigated how the problem of supervised pattern recognition - finding the label to a feature vector given a dataset of labelled feature vectors - can be solved using a universal quantum computer. As a conclusion I want to highlight the central points made in each chapter, summarise the general findings and suggest avenues for further research.

12.1 Summary of findings

Five statements can be derived from the five chapters of Part I and Part II:

Chapter 2. Machine learning algorithms extract decisions from data, and the main computational task is optimisation.
One can understand machine learning methods as computer algorithms that derive predictions or decisions from data, thereby converting large amounts of information into an answer, which in the most extreme case is a simple yes/no decision. For this purpose one defines a family of model functions or model distributions, and adapts the parameters (i.e., weights) or hyperparameters (i.e., the kind of kernel function) to the data. The goal is to find a model that generalises from the data in order to process new instances. This learning phase can often be formulated as an optimisation problem (for example least-squares optimisation or maximum likelihood estimation). An important design principle for the model, optimisation problem and solution method is to avoid overfitting, or learning the particulars of the data rather than the general trends. Approaches like regression, artificial neural networks, graphical models, and kernel methods show a variety of possibilities of fiting models to data.

Chapter 3. Quantum computing offers a range of routines that can be used for optimisation tasks, and they are based on different information encoding strategies.
Quantum computers use binary systems called qubits to process information, and a toolbox of algorithms has been developed that can be used to construct quantum algorithms for supervised pattern recognition. Amongst them are Grover search, quantum phase estimation, postselective amplitude updates, quantum matrix inversion and interference circuits such as the swap test. For machine learning applications, information encoding into quantum systems becomes crucial. Three strategies can be distinguished: encoding classical information into the state of qubits (basis encoding), into the amplitudes of quantum states (amplitude encoding) or into the evolution of

the system (dynamic encoding).

Chapter 4. Efficient state preparation routines to encode large datasets in quantum states are crucial for quantum machine learning algorithms.
Writing large datasets into quantum states is a non-trivial task, but necessarily the first step of a quantum algorithm that performs supervised learning with classical data. This is even more true since the destruction of the final quantum state by measurement forces us to repeat the entire routine for every prediction. Efficient data encoding is particularly difficult for amplitude encoded algorithms, namely if the number of operations is supposed to be sublinear in the dimensions of the dataset to maintain the "super-efficient" runtime of a training algorithm. There are a few proposals that fulfil this requirement, but all place restrictions on the dataset they are able to process.

Chapter 5. While we know that quantum computing can speed up the runtime of classical machine learning algorithms, evidence suggests that it cannot reduce the number of data points needed for learning.
One can distinguish different ways to enhance machine learning with quantum computers. For example, time complexity refers to more efficient algorithms, while the sample complexity compares the number of training vectors needed to learn a model function. Current results suggest that the sample complexity is similar in the quantum and classical setting, while we can hope for speedups in terms of the runtime, which the rest of the thesis focussed on. Model complexity - the flexibility or representational capacity of a 'quantum' model - is the least well understood, as is the robustness of quantum learning against noise.

Chapter 6. The literature on quantum-enhanced machine learning can be distinguished into four different approaches of how to solve optimisation problems for classical models on quantum devices.
The majority of literature on quantum machine learning algorithms on supervised pattern recognition considers a classical method consisting of a distinct model family and a way of how to use the dataset to choose the best model from this family, and translates it into the quantum setting. This means to outsource optimisation problems such as search, matrix inversion, combinatorial optimisation or sampling to a quantum computer, and hope for some speedup in solving the problem. The result of the quantum computation is aimed to reproduce the classical result, but with a faster computation in the light of growing problem sizes.

Part III contributed five quantum machine learning algorithms to tackle the problem of supervised pattern recognition:

Algorithm 1: Quantum least-squares linear regression
The quantum linear regression algorithm implements a singular value decomposition to solve the linear system of equations derived from a convex quadratic optimisation problem. To do this, it combines the tools of quantum matrix exponentiation and the quantum linear systems algorithm. For realistic applications, regularisation terms amend the linear system of equations, and it has been shown that this can be seamlessly integrated into the quantum algorithm. Also, the data matrix that defines the original problem can be replaced by other kernel matrices to effectively

implement a feature map of the data into a higher dimensional space.

Algorithm 2: Quantum gradient descent

In collaboration with Dr Patrick Rebentrost and Prof Seth Lloyd

For more complex optimisation problems, closed form solutions such as linear systems of equations are not possible, and optimisation can be approached by gradient descent methods. In standard gradient descent, the candidate for the solution is iteratively updated by making steps in the direction of steepest descent, while Netwon's method also uses curvature information. The quantum algorithm is based on amplitude encoding, from which it derives its logarithmic runtime in the dimension of the search space. The proposal considers a very specific type of objective function that allows us to compute the derivative and Hessian matrix at the point of the current solution using an extended version of density matrix exponentiation. The probabilistic nature of the quantum algorithm implies that every iteration only produces a proportion of correct copies of the quantum state, and the method therefore scales exponentially with the number of steps. It is therefore suitable for local searches in large search spaces.

Algorithm 3: Quantum nearest neighbour

The quantum version of the nearest neighbour algorithm writes a function of the distance between a new input and the training inputs into the amplitudes of a quantum state. Two versions can be distinguished, depending on if we want to use amplitude encoding or basis encoding for the dataset. The versions implement slightly different distance functions, which can be interpreted as different kernels. The runtime is linear in or even independent from the number of qubits, excluding state preparation. The two algorithms are excellent candidates to implement with small-scale quantum processors due to their simplicity.

Algorithm 4: Quantum perceptron

Quantum neural network models have been extensively referred to in the literature related to quantum machine learning, but rarely manage to translate their basic characteristics - the concatenation of linear algebra and nonlinearities as well as non-convex training - into a quantum setting. Approaching this problem from the perspective of information encoding strategies helps to shed light on the difficulties involved in formulating a quantum neural network that harvests obvious speedups. The quantum perceptron algorithm is a suggestion of how to implement the step activation function on basis encoded inputs by using the quantum phase estimation routine.

Algorithm 5: Quantum ensemble method

Quantum parallelism is an interesting candidate for ensemble methods, where different (quantum) models are combined for prediction. The method defines a quantum ensemble through a state preparation routine of a weighted superposition of base models to which a classification routine can be applied in parallel. This is is close to a Bayesian learning framework. The case of weights that correspond to the accuracy of a model has been further investigated. It was shown that if the model exhibits a certain symmetry, the ensemble will effectively only consist of accurate classifiers. Simulations reveal that such a combined classifier performs well in simple examples.

12.2 Discussion

Quantum machine learning has gained a fair amount of interest in the last number of years - for example as an extension to classical machine learning, as a potential future application for quantum computing, or as a testing ground for quantum annealing devices. Some even hope (or fear) that quantum computing will revolutionise machine learning by making hard training problems solvable, at the same time providing the "killer-app" that the quantum community is waiting for. The insights presented in this thesis show why quantum computing does not offer any miracles or 'free lunches', and for all we know today it is unlikely to be more than an addition to the vast field of classical machine learning research in the near future. It is also not clear that quantum machine learning adds new results in proving the supremacy of quantum computers with exponential speedups. The main argument here is that machine learning does not pose a specific problem to computer science, but combines standard applications such as search, combinatorial optimisation, integration or linear algebra which have already been studied in detail in the past decades of quantum information processing research. Speaking from a complexity theory point of view, quantum machine learning therefore derives its speedups from the parent discipline of quantum computing.

Having said that, and beyond general claims of the power of learning with quantum computers, a quantum perspective to machine learning has a lot to offer for both superordinate disciplines. On the one hand, quantum machine learning algorithms extend the methods of machine learning and provide a different paradigm to think about well-known problems in the field. In practice where approximations can be good alternatives to hard solutions, the more subtle qualities of quantum computing - rather than provable asymptotic runtime speedups - could very well play a role. Furthermore, quantum computing can lead to models widely discarded in classical machine learning to become interesting, as demonstrated with the quantum ensemble method. On the other hand, quantum machine learning challenges the quantum information processing community to develop new tools. Discussed here were variations of the swap test, density matrix exponentiation as well as non-convex methods for optimisation. Another issue raised by the emerging field is data encoding and the classical-quantum interface, which has not been a central point on the agenda of quantum computing. As so often in research, the devil is in the detail, and quantum machine learning certainly has a lot of details worthy to explore.

Another central insight from this thesis is that the main approaches in the literature and presented here have merely touched the possibilities that a truly interdisciplinary approach offers. It has been argued repeatedly that the majority of quantum machine learning algorithms are based on outsourcing problems in the training of textbook models to a quantum computer. In other words, most contributions choose the prototype of a classical machine learning model and try to reproduce it with a quantum algorithm that promises some asymptotic speedup. These models however have been designed for classical computation. To create innovation, quantum machine learning must leave the borders of what has been explored in classical machine learning and design new models suited to quantum devices. This implies a paradigm change. One would have to start with quantum computing and ask what type of models we can realise naturally in this way, after which these models can be investigated and adjusted. In short, we should build genuine "quantum models" for pattern recognition.

An exciting development in quantum computing at the moment is the realisation of small-scale quantum computing devices. At the time of writing, several laboratories have setups of up to ten qubits, usually based on superconducting technology, and promise to scale this up to 50 qubits in the near future. The demand for quantum algorithms that can be implemented in these devices is rising, and this could be a chance for quantum machine learning, as prominently remarked in a recent call in the Nature journal to "commercialize quantum technologies in five years" by leading researchers [242]. The question of what type of models naturally emerge from quantum dynamics has been addressed in the quantum annealing community [188] as well as related to device-oriented approaches to quantum machine learning [243]. It should be an inspiration to quantum algorithms for universal quantum computers.

In summary, changing the approach from mimicking classical results to thinking about quantum models, and from waiting for large-scale quantum computers to considering small-scale devices is necessary to turn the decidedly theoretical nature of quantum machine learning into a field that matches the practical motivation of classical machine learning.

12.3 Suggestions for further research

A number of future research directions emerge from questions raised in the main body of the thesis, as well as the previous discussion. First, as argued above the creation of new models that are suitable to quantum information processing is a promising avenue of research. Second, the development and implementation of quantum machine learning algorithms for small-scale quantum computing devices would allow us to discover the practical problems of quantum machine learning and make progress in benchmarking for practical applications. State preparation as well as the core machine learning routines would have to be decomposed into low-depth sequences of elementary quantum gates and adapted to the hardware. Third, a lot more research is necessary in the area of state preparation and the interface between classical data and quantum computers. This is particularly important if amplitude encoding is used to claim exponential speedups. Possible projects could develop quantum circuits that encode typical datasets into amplitudes, or determine which structures in data allows state preparation routines that only grow polynomially with the number of qubits. Fourth, it is crucial to develop the field of quantum optimisation with respect to machine learning problems. Just as classical machine learning poses new challenges to the mathematical theory of optimisation, we would need to develop more complex methods of quantum optimisation that can fit models to data. Non-convex and nonlinear optimisation problems with very little knowledge about the structure of the landscape in combination with regularisation techniques would be of particular interest.

To approach these open problems, quantum machine learning would hugely benefit from the active cooperation between machine learning experts and quantum information scientists, as well as between theory and application. In other words, the discipline has to be released from the grip of mainly theoretical quantum computing research and opened to a broader range of expertise. This requires some effort in cross-disciplinary communication about a technically very demanding subject matter. Overall, the potential to grow quantum machine learning into a substantial research discipline is certainly there.

Appendix A

Density matrix exponentiation: Proofs

A.1 Basic density matrix exponentiation

What follows is the proof of the equation

$$\mathrm{tr}_2\left[e^{-iS\Delta t}(\sigma \otimes \rho)e^{iS\Delta t}\right] = e^{-iH_\rho \Delta t} + \mathcal{O}(\Delta t^2),$$

where S swaps the two states σ and ρ and gets simulated for time Δt. With this equation, Lloyd et al. [140] show that by simulating a swap operator and tracing out the second density matrix, one effectively simulates a Hamiltonian that corresponds to the second density matrix.

Consider two general mixed states $\rho = \sum_{ii'=1}^{N} a_{ii'}|i\rangle\langle i'|$ and $\sigma = \sum_{jj'=1}^{N} b_{jj'}|j\rangle\langle j'|$:

$$
\begin{aligned}
&\mathrm{tr}_2\big[e^{-iS\Delta t}(\sigma\otimes\rho)e^{iS\Delta t}\big] \\
&= \mathrm{tr}_2\big[\sum_{k,k'=0}^{\infty}\frac{(-i)^k\Delta t^k i^{k'}\Delta t^{k'}}{k!k'!}\ S^k(\sigma\otimes\rho)S^{k'}\big] \\
&= \mathrm{tr}_2\big[(\sigma\otimes\rho)+i\Delta t\ \not{\not{H}}(\sigma\otimes\rho)S - i\Delta t\ \ S(\sigma\otimes\rho)\not{\not{H}} + \mathcal{O}(\Delta t^2)\big] \\
&= \mathrm{tr}_2\big[(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|) + i\Delta t(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|)S \\
&\quad - i\Delta t S(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|) + \mathcal{O}(\Delta t^2)\big] \\
&= \mathrm{tr}_2\big[(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'|\otimes|i\rangle\langle i'|) + i\Delta t(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle i'|\otimes|i\rangle\langle j'|) \\
&\quad - i\Delta t(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|i\rangle\langle j'|\otimes|j\rangle\langle i'|) + \mathcal{O}(\Delta t^2)\big] \\
&= \sum_{jj'}b_{jj'}|j\rangle\langle j'| + i\Delta t\sum_{ij}\sum_{k}a_{ki}b_{jk}|j\rangle\langle i| - i\Delta t\sum_{ij}\sum_{k}a_{ik}b_{kj}|i\rangle\langle j| + \mathcal{O}(\Delta t^2) \\
&= \sigma + i\Delta t(\sum_{ij}\sum_{k}a_{ki}b_{jk}|j\rangle\langle i| - \sum_{ij}\sum_{k}a_{ik}b_{kj}|i\rangle\langle j|) + \mathcal{O}(\Delta t^2) \\
&= \sigma + i\Delta t(\sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|j\rangle\langle j'||i\rangle\langle i'| - \sum_{ii'}\sum_{jj'}a_{ii'}b_{jj'}|i\rangle\langle i'||j\rangle\langle j'|) + \mathcal{O}(\Delta t^2) \\
&= \sigma + i\Delta t(\sigma\rho-\rho\sigma) + \mathcal{O}(\Delta t^2) \\
&= \sigma - i\Delta t[\rho,\sigma] + \mathcal{O}(\Delta t^2)
\end{aligned}
$$

Therefore, the circuit simulates $e^{-i\rho\Delta t}\sigma e^{i\rho\Delta t}$, if only the time steps are sufficiently short.

A.2 Density matrix exponentiation in superposition

In order to extract the eigenvalues of a density operator, one has to extend the previous routine in order to simulate $\sum_{n=1}^{N}|n\Delta t\rangle\langle n\Delta t|\otimes e^{-i\rho n\Delta t}$ 'in superposition'. This can be done by taking N copies of ρ.

$$
\begin{aligned}
&\mathrm{tr}_{1..N}\Big\{\tfrac{1}{2^q}\sum_{n=1}^{N}|n\Delta t\rangle\langle n\Delta t|\otimes\prod_{g=1}^{n}e^{-iS_g\Delta t}\Big(\sum_{n''}|n''\Delta t\rangle\langle n''\Delta t|\otimes\sigma\otimes\underbrace{\rho^{(1)}\otimes\ldots\otimes\rho^{(N)}}_{\rho_{\mathrm{tot}}}\Big)\sum_{n'=1}^{N}|n'\Delta t\rangle\langle n'\Delta t|\otimes\prod_{g'=1}^{n'}e^{iS_{g'}\Delta t}\Big\} \\
&= \mathrm{tr}_{1..N}\Big\{\sum_{n=1}^{N}|n\Delta t\rangle\langle n\Delta t|\otimes\prod_{g=1}^{n}e^{-iS_g\Delta t}\Big(\sigma\otimes\rho^{(1)}\otimes\ldots\otimes\rho^{(N)}\Big)\prod_{g'=1}^{n}e^{iS_{g'}\Delta t}\Big\} \\
&= \mathrm{tr}_{1..N}\Big\{\sum_{n=1}^{N}|n\Delta t\rangle\langle n\Delta t|\otimes\Big(\sigma\otimes\rho_{tot}-i\Delta t\sum_{g=1}^{n+1}\Big((\sigma\otimes\rho^{(g)})S-S(\sigma\otimes\rho^{(g)})\Big)\otimes\bigotimes_{i\neq g}\rho^{(i)}+O(\Delta t^2)\Big)\Big\} \\
&= \sum_{n=1}^{N}|n\Delta t\rangle\langle n\Delta t|\otimes\Big(\sigma-i\Delta t\sum_{g=1}^{n+1}[\rho^{(g)},\sigma]+O(\Delta t^2)\Big)
\end{aligned}
$$

If all ρ are the same,

$$\sum_{n=1}^{N} |n\Delta t\rangle\langle n\Delta t| \otimes \Big(\sigma - i\Delta t[n\rho, \sigma] + O(\Delta t^2)\Big),$$

which corresponds to

$$\sum_{n=1}^{N} |n\Delta t\rangle\langle n\Delta t| \otimes e^{-i\rho n\Delta t}\sigma e^{i\rho n\Delta t}$$

up to terms $\mathcal{O}(\Delta t^2)$.

Above I used that

$$\begin{aligned}
&\prod_{g=1}^{2} e^{-iS_g\Delta t}\Big(\sigma\otimes\rho^{(1)}\otimes\ldots\otimes\rho^{(N)}\Big)\prod_{g'=1}^{2} e^{iS_{g'}\Delta t}\\
&= e^{-iS_2\Delta t}\Big(\sigma\otimes\rho_{tot} - i\Delta t\Big((\sigma\otimes\rho^{(1)})S - S(\sigma\otimes\rho^{(1)})\Big)\otimes\bigotimes_{i=2}^{N}\rho^{(i)}\Big)e^{iS_2\Delta t} + O(\Delta t^2)\\
&= e^{-iS_2\Delta t}\Big(\sigma\otimes\rho_{tot}\Big)e^{iS_2\Delta t} - i\Delta t e^{-iS_2\Delta t}\Big((\sigma\otimes\rho^{(1)})S - S(\sigma\otimes\rho^{(1)})\Big)\otimes\bigotimes_{i=2}^{N}\rho^{(i)}\Big)e^{iS_2\Delta t} + O(\Delta t^2)\\
&= \sigma\otimes\rho_{tot} - i\Delta t\Big((\sigma\otimes\rho^{(2)})S - S(\sigma\otimes\rho^{(2)})\Big)\otimes\bigotimes_{i\neq 2}\rho^{(i)} - i\Delta t\Big((\sigma\otimes\rho^{(1)})S - S(\sigma\otimes\rho^{(1)})\Big)\otimes\bigotimes_{i\neq 2}\rho^{(i)} + O(\Delta t^2),
\end{aligned}$$

and

$$\begin{aligned}
&e^{-iS_{n+1}\Delta t}\Big(\sigma\otimes\rho_{tot} - i\Delta t\sum_{g=1}^{n}\Big((\sigma\otimes\rho^{(g)})S - S(\sigma\otimes\rho^{(g)})\Big)\otimes\bigotimes_{i\neq g}\rho^{(i)} + O(\Delta t^2)\Big)e^{iS_{n+1}\Delta t}\\
&= \sigma\otimes\rho_{tot} - i\Delta t\sum_{g=1}^{n+1}\Big((\sigma\otimes\rho^{(g)})S - S(\sigma\otimes\rho^{(g)})\Big)\otimes\bigotimes_{i\neq g}\rho^{(i)} + O(\Delta t^2).
\end{aligned}$$

Appendix B

Quantum gradient descent: Proofs

B.1 Local convergence proof for projected gradient descent

This work was done together with Dr. Patrick Rebentrost from the Massachussetts Institute of Technology.

We provide a simple proof of convergence for the projected gradient descent method in the limit of small step sizes in the vicinity of a minimum of a locally convex function for which, without loss of generality, $o(|\boldsymbol{\theta}^{(t)}\rangle) > 0$. Let the unnormalized gradient step be

$$|\phi\rangle = |\boldsymbol{\theta}^{(t)}\rangle - \eta|\nabla o(\boldsymbol{\theta}^{(t)})\rangle. \tag{B.1}$$

The projected (normalized) gradient step is

$$|\mathbf{x}^{(t+1)}\rangle = \frac{1}{C^{(t+1)}}\left(|\boldsymbol{\theta}^{(t)}\rangle - \eta|\nabla o(\boldsymbol{\theta}^{(t)})\rangle\right), \tag{B.2}$$

with the normalization factor given by $(C^{(t+1)})^2 = 1 - 2\eta\langle\mathbf{x}^{(t)}|D|\mathbf{x}^{(t)}\rangle + \eta^2\langle\mathbf{x}^{(t)}|D^2|\mathbf{x}^{(t)}\rangle$. For the unnormalized state there exists $\eta_0 > 0$ such that for all small $\eta < \eta_0$ we have

$$\eta\langle\nabla o(\boldsymbol{\theta}^{(t)})|\nabla o(\boldsymbol{\theta}^{(t)})\rangle + \mathcal{O}(\eta^2) = o(|\boldsymbol{\theta}^{(t)}\rangle) - o(|\phi\rangle) \geq 0. \tag{B.3}$$

For the normalized state:

$$\begin{aligned}
o(|\boldsymbol{\theta}^{(t)}\rangle) - o(|\mathbf{x}^{(t+1)}\rangle) &= o(|\boldsymbol{\theta}^{(t)}\rangle) - \frac{1}{(C^{(t+1)})^{2p}}o(|\phi\rangle) \qquad \text{(B.4)}\\
&= o(|\boldsymbol{\theta}^{(t)}\rangle) - \frac{1}{(C^{(t+1)})^{2p}}\left[o(|\boldsymbol{\theta}^{(t)}\rangle) - \eta\langle\nabla o(\boldsymbol{\theta}^{(t)})|\nabla o(\boldsymbol{\theta}^{(t)})\rangle + \mathcal{O}(\eta^2)\right].\\
&= o(|\boldsymbol{\theta}^{(t)}\rangle) - \left(1 + 4\eta p\langle\boldsymbol{\theta}^{(t)}|\nabla o(\boldsymbol{\theta}^{(t)})\rangle + \mathcal{O}(\eta^2)\right)\\
&\quad \left[o(|\boldsymbol{\theta}^{(t)}\rangle) - \eta\langle\nabla o(\boldsymbol{\theta}^{(t)})|\nabla o(\boldsymbol{\theta}^{(t)})\rangle + \mathcal{O}(\eta^2)\right].\\
&= \eta\langle\nabla o(\boldsymbol{\theta}^{(t)})|\nabla o(\boldsymbol{\theta}^{(t)})\rangle - 4\eta p\langle\boldsymbol{\theta}^{(t)}|\nabla o(\boldsymbol{\theta}^{(t)})\rangle o(|\boldsymbol{\theta}^{(t)}\rangle) + \mathcal{O}(\eta^2).
\end{aligned}$$

Thus, we obtain the condition

$$\langle\boldsymbol{\theta}^{(t)}|\nabla o(\boldsymbol{\theta}^{(t)})\rangle \leq \frac{\langle\nabla o(\boldsymbol{\theta}^{(t)})|\nabla o(\boldsymbol{\theta}^{(t)})\rangle}{4po(|\boldsymbol{\theta}^{(t)}\rangle)}. \tag{B.5}$$

Hence we expect our projected method to work well if $\langle \boldsymbol{\theta}^{(t)} | \nabla o(\boldsymbol{\theta}^{(t)}) \rangle \ll 1$, which means the gradient has a large component orthogonal to the current solution. Intuitively that means that the gradient points tangential to the unit ball. If the current solution and gradient are parallel, we cannot optimize further on the unit ball and our norm-constrained problem is solved.

B.2 Quantum Newton's method: Details

The matrix M_{H_A} corresponding to the Hessian part H_A, defined in Eq. (8.7), used for the sparse matrix simulation methods is given by

$$M_{H_A} = \frac{1}{2} \sum_{\alpha} \sum_{j \neq k} \left(\prod_{i \neq j,k} \mathbf{A}_i^{\alpha} \right) \otimes \left[\left(\mathcal{I} \otimes ((\mathbf{A}_k^{\alpha})^T + \mathbf{A}_k^{\alpha}) \right) S \left(\mathcal{I} \otimes ((\mathbf{A}_j^{\alpha})^T + \mathbf{A}_j^{\alpha}) \right) \right]. \quad \text{(B.6)}$$

Here, S is the swap matrix between the last two registers. The operator M_{H_A} is sparse if the constituent matrices A_j^{α} are sparse. We assume that the matrices $\mathbf{A}_j^{\alpha}$ are given via an oracle. The operator H_A is obtained from

$$H_A = \mathrm{tr}_{1 \ldots p-1} \{ \rho^{\otimes (p-1)} \, M_{H_A} \}, \quad \text{(B.7)}$$

as before. In case M_{H_A} is non-symmetric, we can define a symmetric matrix analogous to Eq. (8.9).

The quantum Newton step corresponds goes as follows. Assume we have prepared the gradient descent step up to the state:

$$|\psi\rangle = \frac{1}{\sqrt{p_D}} \left(\cos\gamma \, |0\rangle |\psi_{\boldsymbol{\theta}}\rangle + \mathrm{i} \, C_D \sin\gamma \, |1\rangle D |\psi_{\boldsymbol{\theta}}\rangle \right). \quad \text{(B.8)}$$

The eigenstates of H are here given by $\{|\chi_l\rangle\}$ and the eigenvalues by $\{\lambda_l\}$. First, perform phase estimation with H conditioned on the ancilla being $|1\rangle$ to arrive at

$$\frac{1}{\sqrt{p_D}} \left(\cos\gamma \, |0\rangle |\psi_{\boldsymbol{\theta}}\rangle |0\rangle |1\rangle + \mathrm{i} \, C_D \sin\gamma \, |1\rangle \sum_l \beta_l |\chi_l\rangle |\lambda_l\rangle \right). \quad \text{(B.9)}$$

Perform a conditional rotation of another ancilla and uncompute the eigenvalue register:

$$\frac{1}{\sqrt{p_D}} \left(\cos\gamma \, |0\rangle |\psi_{\boldsymbol{\theta}}\rangle |1\rangle + \mathrm{i} \, C_D \sin\gamma \, |1\rangle \sum_l \beta_l |\chi_l\rangle \left(\sqrt{1 - \left(\frac{C_H}{\lambda_l} \right)^2} \, |0\rangle + \frac{C_H}{\lambda_l} |1\rangle \right) \right). \quad \text{(B.10)}$$

Here, $C_H = \mathcal{O}(1/\kappa_H)$, where κ_H is the condition number of H. Measurement of the ancilla in $|1\rangle$ arrives at

$$\frac{1}{\sqrt{p_H}} \left(\cos\gamma \, |0\rangle |\psi_{\boldsymbol{\theta}}\rangle + \mathrm{i} \, C_D C_H \sin\gamma \, |1\rangle \sum_l \frac{1}{\lambda_l} \beta_l |\chi_l\rangle \right), \quad \text{(B.11)}$$

which is the desired state

$$\frac{1}{\sqrt{p_H}} \left(\cos\gamma \, |0\rangle |\psi_{\boldsymbol{\theta}}\rangle + \mathrm{i} \, C_D C_H \sin\gamma \, |1\rangle H^{-1} D |\psi_{\boldsymbol{\theta}}\rangle \right). \quad \text{(B.12)}$$

The success probability of the measurement is given by:

$$p_H = \cos^2\gamma + \sin^2\gamma\, C_D^2 C_H^2 \langle\psi_{\boldsymbol{\theta}}|DH^{-2}D|\psi_{\boldsymbol{\theta}}\rangle. \tag{B.13}$$

B.3 Quantum principal component analysis with erroneous states

The simulation of the operators D and H incurs additional errors because the current solution $|\psi_{\boldsymbol{\theta}}\rangle$ used to construct the operators is given to an error. We provide a simple argument for an error averaging at the cost of number of copies required for the simulation. First note that

$$D = \mathrm{tr}_{1\ldots p-1}\{\rho^{\otimes(p-1)}\ M_D\}$$

with $\rho = |\psi_{\boldsymbol{\theta}}\rangle\langle\psi_{\boldsymbol{\theta}}|$. Thus, if $|\psi_{\boldsymbol{\theta}}\rangle$ is given to an error ϵ_0 then D is given to an error $\mathcal{O}(p\epsilon_0)$ and the same for H.
For the error discussion, assume for simplicity a simple binomial distribution, where errors come as $+p\epsilon_0\tilde{D}$ and $-p\epsilon_0\tilde{D}$, where $\tilde{D}$ is some operator with $\|\tilde{D}\| \leq 1$. The argument holds also for other distributions. Two simulation steps with opposite signs are given by

$$\begin{aligned} e^{\mathrm{i}\,(D+p\epsilon_0\tilde{D})\Delta t}e^{\mathrm{i}\,(D-p\epsilon_0\tilde{D})\Delta t} &= (\mathcal{I} + \mathrm{i}\,(D+p\epsilon_0\tilde{D})\Delta t + \mathcal{O}(\Delta t^2))(\mathcal{I} + \mathrm{i}\,(D-p\epsilon_0\tilde{D})\Delta t + \mathcal{O}(\Delta t^2)) \\ &= (\mathcal{I} + \mathrm{i}\,D\Delta t)^2 + \mathcal{O}(\Delta t^2). \end{aligned} \tag{B.14}$$

The task is to simulate $e^{-\mathrm{i}\,Dt}$ for a time $t = m\Delta t$ to obtain an overall error $\epsilon > 0$. Note that after m steps and averaging over the signs, we have:

$$\|e^{\mathrm{i}\,(D\pm p\epsilon_0\tilde{D})\Delta t}\cdots e^{\mathrm{i}\,(D\pm p\epsilon_0\tilde{D})\Delta t} - e^{\mathrm{i}\,Dt}\| = \mathcal{O}(\sqrt{m}p\epsilon_0\Delta t + m\Delta t^2) := \epsilon. \tag{B.15}$$

The error is given by:

$$\epsilon = \mathcal{O}(\sqrt{m}p\epsilon_0\Delta t + m\Delta t^2) = \mathcal{O}\left(\frac{p\epsilon_0 t}{\sqrt{m}} + \frac{t^2}{m}\right). \tag{B.16}$$

Thus, solving for the number of steps m leads to

$$m = \mathcal{O}\left(\frac{p^2\epsilon_0^2 t^2}{\epsilon^2} + \frac{t^2}{\epsilon}\right) = \mathcal{O}\left(\left(1 + \frac{p^2\epsilon_0^2}{\epsilon}\right)\frac{t^2}{\epsilon}\right). \tag{B.17}$$

This compares to $m = \mathcal{O}\left(\frac{t^2}{\epsilon}\right)$ for the usual QPCA algorithm assuming no errors in the state encoding the Hamiltonian. Since we run the algorithm for a time $t = 1/\epsilon$, we obtain a total number of copies

$$m = \mathcal{O}\left(\left(1 + \frac{p^2\epsilon_0^2}{\epsilon}\right)\frac{1}{\epsilon^3}\right). \tag{B.18}$$

Bibliography

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

[2] Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J Briegel. Quantum speedup for active learning agents. *Physical Review X*, 4(3):031002, 2014.

[3] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Machine learning in a quantum world. In *Advances in Artificial Intelligence*, pages 431–442. Springer, 2006.

[4] Vedran Dunjko, Jacob M Taylor, and Hans J Briegel. Quantum-enhanced machine learning. *Physical Review Letters*, 117(13):130501, 2016.

[5] Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.

[6] Federico Holik, Giuseppe Sergioli, Hector Freytes, and Angelo Plastino. Pattern recognition in non-Kolmogorovian structures. *arXiv preprint arXiv:1609.06340*, 2016.

[7] Masahide Sasaki and Alberto Carlini. Quantum learning and universal quantum matching machine. *Physical Review A*, 66(2):022303, 2002.

[8] Juan Carrasquilla and Roger G Melko. Machine learning phases of matter. *arXiv preprint arXiv:1605.01735*, 2016.

[9] Alessandro Bisio, Giulio Chiribella, Giacomo Mauro DAriano, Stefano Facchini, and Paolo Perinotti. Optimal quantum learning of a unitary transformation. *Physical Review A*, 81(3):032324, 2010.

[10] G Sentís, E Bagan, J Calsamiglia, and R Munoz-Tapia. Multicopy programmable discrimination of general qubit states. *Physical Review A*, 82(4):042312, 2010.

[11] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area V2. In *Advances in Neural Information Processing Systems*, pages 873–880, 2008.

[12] Christopher M Bishop. *Pattern recognition and machine learning*, volume 1. Springer, 2006.

[13] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, New York, 2012.

[14] Kevin P. Murphy. *Machine Learning. A probabilistic perspective*. MIT Press, 2012.

[15] Daphne Koller and Nir Friedman. *Probabilistic graphical models: Principles and techniques*. MIT Press, 2009.

[16] John A Hertz, Anders S Krogh, and Richard G Palmer. *Introduction to the theory of neural computation*, volume 1. Westview Press, Redwood City (California), 1991.

[17] Christopher M Bishop. *Neural Networks for Pattern Recognition*, volume 1. Clarendon Press, Oxford, 1995.

[18] Ethem Alpaydin. *Introduction to machine learning*. MIT Press, Cambridge, 2004.

[19] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer, Berlin, 2001.

[20] Alex Monràs, Gael Sentís, and Peter Wittek. Inductive quantum learning: Why you are doing it almost right. *arXiv preprint arXiv:1605.07541*, 2016.

[21] James D Malley and John Hornstein. Quantum statistical inference. *Statistical Science*, pages 433–457, 1993.

[22] Kristin P Bennett and Emilio Parrado-Hernández. The interplay of optimization and machine learning research. *Journal of Machine Learning Research*, 7(Jul):1265–1281, 2006.

[23] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[24] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[25] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[26] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[27] Stephen A Vavasis. *Nonlinear optimization: Complexity issues*. Oxford University Press, Inc., 1991.

[28] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, 2001.

[29] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The Annals of Statistics*, pages 1171–1220, 2008.

[30] Carl Edward Rasmussen. *Gaussian processes for machine learning*. Citeseer, 2006.

[31] Thomas Griffiths and Alan Yuille. A primer on probabilistic inference. *The Probabilistic Mind: Prospects for Bayesian cognitive science*, pages 33–57, 2008.

[32] Andrew Y Ng and A Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in Neural Information Processing Systems*, 14:841–846, 2002.

[33] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[34] Karl Weierstrass. Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639, 1885.

[35] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, 1943.

[36] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[37] Marvin Minsky and Seymour Papert. *Perceptrons: An introduction to computational geometry*. MIT Press, Cambridge, 1969.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.

[39] Albert BJ Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

[40] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1310–1318. IEEE, 2013.

[42] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[43] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems*, volume 400, page 409, 1995.

[44] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, 2 edition, 2000.

[45] Geoffrey E Hinton and Terrence J Sejnowski. Optimal perceptual inference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 448–453. Citeseer, 1983.

[46] Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[47] Geoffrey Hinton. A practical guide to training restricted Boltzmann machines, 2010. UTML TR 2010-003, Version 1.

[48] Miguel A Carreira-Perpinan and Geoffrey Hinton. On contrastive divergence learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 10, pages 33–40. Citeseer, 2005.

[49] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[50] Ilya Sutskever and Tijmen Tieleman. On the convergence properties of contrastive divergence. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 789–795, 2010.

[51] Judea Pearl. *Causality*. Cambridge University Press, 2009.

[52] David Heckerman, Dan Geiger, and David M Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[53] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[54] Irad Ben-Gal. Bayesian networks. *Encyclopedia of Statistics in Quality and Reliability*, 2007.

[55] Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[56] Sahibsingh A Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, 4:325–327, 1976.

[57] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.

[58] Matthew S Leifer and David Poulin. Quantum graphical models and belief propagation. *Annals of Physics*, 323(8):1899–1946, 2008.

[59] Scott Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013.

[60] Andrew Whitaker. *Einstein, Bohr and the quantum dilemma: From quantum theory to quantum information*. Cambridge University Press, 2006.

[61] Albert Einstein. Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Annalen der Physik*, 322(8):549–560, 1905.

[62] Albert Einstein. Zur elektrodynamik bewegter Körper. *Annalen der Physik*, 322(10):891–921, 1905.

[63] Misha Denil and Nando De Freitas. Toward the implementation of a quantum RBM. In *Proceedings of the NIPS Deep Learning and Unsupervised Feature Learning Workshop*, 2011.

[64] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996.

[65] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 439, pages 553–558. The Royal Society, 1992.

[66] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[67] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117. The Royal Society, 1985.

[68] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2010.

[69] International Business Machines. Ibm quantum experience.

[70] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[71] John Clarke and Frank K Wilhelm. Superconducting quantum bits. *Nature*, 453(7198):1031–1042, 2008.

[72] Pieter Kok, William J Munro, Kae Nemoto, Timothy C Ralph, Jonathan P Dowling, and Gerard J Milburn. Linear optical quantum computing with photonic qubits. *Reviews of Modern Physics*, 79(1):135, 2007.

[73] Juan I Cirac and Peter Zoller. Quantum computations with cold trapped ions. *Physical Review Letters*, 74(20):4091, 1995.

[74] Chetan Nayak, Steven H Simon, Ady Stern, Michael Freedman, and Sankar Das Sarma. Non-Abelian anyons and topological quantum computation. *Reviews of Modern Physics*, 80(3):1083, 2008.

[75] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000. MIT-CTP-2936.

[76] Arnab Das and Bikas K Chakrabarti. Colloquium: Quantum annealing and analog quantum computation. *Reviews of Modern Physics*, 80(3):1061, 2008.

[77] Bettina Heim, Troels F Rønnow, Sergei V Isakov, and Matthias Troyer. Quantum versus classical annealing of Ising spin glasses. *Science*, 348(6231):215–217, 2015.

[78] Giuseppe E Santoro and Erio Tosatti. Optimization using quantum mechanics: Quantum annealing through adiabatic evolution. *Journal of Physics A*, 39(36):R393, 2006.

[79] Hans J Briegel and Robert Raussendorf. Persistent entanglement in arrays of interacting particles. *Physical Review Letters*, 86(5):910, 2001.

[80] Michael A Nielsen. Cluster-state quantum computation. *Reports on Mathematical Physics*, 57(1):147–161, 2006.

[81] Robert Prevedel, Andr Stefanov, Philip Walther, and Anton Zeilinger. Experimental realization of a quantum game on a one-way quantum computer. *New Journal of Physics*, 9(6):205, 2007.

[82] Sang Min Lee, Hee Su Park, Jaeyoon Cho, Yoonshik Kang, Jae Yong Lee, Heonoh Kim, Dong-Hoon Lee, and Sang-Kyung Choi. Experimental realization of a four-photon seven-qubit graph state for one-way quantum computation. *Optics Express*, 20(7):6915–6926, Mar 2012.

[83] Mark S. Tame, Robert Prevedel, Mauro Paternostro, Pascal Böhi, M. S. Kim, and Anton Zeilinger. Experimental realization of Deutsch's algorithm in a one-way quantum computer. *Physical Review Letters*, 98:140501, 2007.

[84] Mark S. Tame, Bryn A. Bell, Carlo Di Franco, William J. Wadsworth, and John G. Rarity. Experimental realization of a one-way quantum computer algorithm solving Simon's problem. *Physical Review Letters*, 113:200501, Nov 2014.

[85] Charles H Bennett. Logical reversibility of computation. *Maxwell's Demon. Entropy, Information, Computing*, pages 197–204, 1973.

[86] Nicolas Gisin. Weinberg's non-linear quantum mechanics and supraluminal communications. *Physics Letters A*, 143(1):1–2, 1990.

[87] Joseph Polchinski. Weinberg's nonlinear quantum mechanics and the Einstein-Podolsky-Rosen paradox. *Physical Review Letters*, 66:397–400, 1991.

[88] Daniel S Abrams and Seth Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and # P problems. *Physical Review Letters*, 81(18):3992, 1998.

[89] Asher Peres. Nonlinear variants of Schrödinger's equation violate the second law of thermodynamics. *Physical Review Letters*, 63(10):1114, 1989.

[90] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009.

[91] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *arXiv preprint quant-ph/9605034*, 1996.

[92] Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*, pages 636–643. ACM, 2000.

[93] Eli Biham, Ofer Biham, David Biron, Markus Grassl, and Daniel A Lidar. Grovers quantum search algorithm for an arbitrary initial amplitude distribution. *Physical Review A*, 60(4):2742, 1999.

[94] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *arXiv preprint quant-ph/0005055*, 2000.

[95] John Watrous. Theory of quantum information, 2011. Lecture Notes from the University of Waterloo, Fall 2011.

[96] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

[97] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM Symposium on Theory of Computing*, pages 333–342. ACM, 2011.

[98] Maris Ozols, Martin Roetteler, and Jérémie Roland. Quantum rejection sampling. *ACM Transactions on Computation Theory (TOCT)*, 5(3):11, 2013.

[99] Daniel S Abrams and Seth Lloyd. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Physical Review Letters*, 83(24):5162, 1999.

[100] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 454, pages 339–354. The Royal Society, 1998.

[101] Hirotada Kobayashi, Keiji Matsumoto, and Tomoyuki Yamakami. Quantum Merlin-Arthur proof systems: Are multiple Merlins more helpful to Arthur? In *Algorithms and Computation*, pages 189–198. Springer, 2003.

[102] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[103] Richard Bonner and Rusinš Freivalds. A survey of quantum learning. In *Proceedings of the Workshop on Quantum Computation and Learning*, page 106, 2003.

[104] Rocco A Servedio and Steven J Gortler. Equivalences and separations between quantum and classical learnability. *SIAM Journal on Computing*, 33(5):1067–1092, 2004.

[105] Hartmut Neven, Vasil S Denchev, Geordie Rose, and William G Macready. Training a binary classifier with the quantum adiabatic algorithm. *arXiv preprint arXiv:0811.0416*, 2008.

[106] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013.

[107] Peter Wittek. *Quantum machine learning: What quantum computing means to data mining*. Academic Press, 2014.

[108] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.

[109] Masahide Sasaki, Alberto Carlini, and Richard Jozsa. Quantum template matching. *Physical Review A*, 64(2):022317, 2001.

[110] Alexander Hentschel and Barry C Sanders. Machine learning for precise quantum measurement. *Physical Review Letters*, 104(6):063603, 2010.

[111] G Sentís, J Calsamiglia, Ramón Muñoz-Tapia, and E Bagan. Quantum learning without quantum memory. *Scientific Reports*, 2(708):1–8, 2012.

[112] Scott Aaronson. The learnability of quantum states. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 463, pages 3089–3114. The Royal Society, 2007.

[113] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *arXiv preprint arXiv:1611.09347*, 2016.

[114] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing*, pages 20–29. ACM, 2003.

[115] Francesco Petruccione Maria Schuld. Quantum machine learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopaedia of Machine Learning and Data Mining*. Springer, 2016.

[116] Dan Ventura and Tony Martinez. Quantum associative memory. *Information Sciences*, 124(1):273–296, 2000.

[117] Carlo A Trugenberger. Probabilistic quantum memories. *Physical Review Letters*, 87:067901, Jul 2001.

[118] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16):160501, 2008.

[119] Simone Raoux, Geoffrey W Burr, Matthew J Breitwisch, Charles T Rettner, Y-C Chen, Robert M Shelby, Martin Salinga, Daniel Krebs, S-H Chen, H-L Lung, et al. Phase-change random access memory: A scalable technology. *IBM Journal of Research and Development*, 52(4.5):465–479, 2008.

[120] Thi Ha Kyaw, Simone Felicetti, Guillermo Romero, Enrique Solano, and Leong-Chuan Kwek. Scalable quantum memory in the ultrastrong coupling regime. *Scientific Reports*, 5(8621), 2015.

[121] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015.

[122] M Kliesch, T Barthel, C Gogolin, M Kastoryano, and J Eisert. Dissipative quantum Church-Turing theorem. *Physical Review Letters*, 107(12):120501, 2011.

[123] Phillip Kaye and Michele Mosca. Quantum networks for generating arbitrary quantum states. In *Proceedings of the International Conference on Quantum Information*. ICQI, 2001. preprint quant-ph/0407102v1.

[124] Andrei N Soklakov and Rüdiger Schack. Efficient state preparation for a register of quantum bits. *Physical Review A*, 73(1):012307, 2006.

[125] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint arXiv:0208112v1*, 2002.

[126] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physcial Review Letters*, 113:130503, Sep 2014.

[127] Anupam Prakash. *Quantum Algorithms for Linear Algebra and Machine Learning*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2014.

[128] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical Review Letters*, 109(5):050505, 2012.

[129] B David Clader, Bryan C Jacobs, and Chad R Sprouse. Preconditioned quantum linear system algorithm. *Physical Review Letters*, 110(25):250504, 2013.

[130] Zhikuan Zhao, Jack K Fitzsimons, and Joseph F Fitzsimons. Quantum assisted Gaussian process regression. *arXiv preprint arXiv:1512.03929*, 2015.

[131] Barbara Kraus, HP Büchler, S Diehl, A Kantian, A Micheli, and P Zoller. Preparation of entangled states by quantum Markov processes. *Physical Review A*, 78(4):042307, 2008.

[132] David Gross, Yi-Kai Liu, Steven T Flammia, Stephen Becker, and Jens Eisert. Quantum state tomography via compressed sensing. *Physical Review Letters*, 105(15):150401, 2010.

[133] Iulia M. Georgescu, Sahel Ashhab, and Franco Nori. Quantum simulation. *Review of Modern Physics*, 86:153–185, 2014.

[134] Dominic W Berry, Graeme Ahokas, Richard Cleve, and Barry C Sanders. Efficient quantum algorithms for simulating sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, 2007.

[135] Dominic W Berry, Andrew M Childs, Richard Cleve, Robin Kothari, and Rolando D Somma. Exponential improvement in precision for simulating sparse Hamiltonians. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 283–292. ACM, 2014.

[136] Dominic W Berry, Andrew M Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 792–809. IEEE, 2015.

[137] Andrew M Childs and Robin Kothari. Limitations on the simulation of non-sparse Hamiltonians. *Quantum Information & Computation*, 10(7):669–684, 2010.

[138] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073, 1996.

[139] Anmer Daskin and Sabre Kais. Decomposition of unitary matrices for finding quantum circuits: Application to molecular Hamiltonians. *The Journal of Chemical Physics*, 134(14):144112, 2011.

[140] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10:631–633, 2014.

[141] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[142] John Watrous. Quantum computational complexity. In *Encyclopedia of complexity and systems science*, pages 7174–7201. Springer, 2009.

[143] Troels F. Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V. Isakov, David Wecker, John M. Martinis, Daniel A. Lidar, and Matthias Troyer. Defining and detecting quantum speedup. *Science*, 345:420–424, 2014.

[144] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

[145] Damian S Steiger and Matthias Troyer. Racing in parallel: Quantum versus classical. In *APS Meeting Abstracts*, 2016.

[146] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[147] Ashish Kapoor, Nathan Wiebe, and Krysta Svore. Quantum perceptron models. In *Advances in Neural Information Processing Systems*, pages 3999–4007, 2016.

[148] Theodore J. Yoder Guang Hao Low and Isaac L. Chuang. Quantum inference on Bayesian networks. *Physical Review A*, 89:062315, 2014.

[149] Hartmut Neven, Vasil S Denchev, Geordie Rose, and William G Macready. Training a large scale classifier with the quantum adiabatic algorithm. *arXiv preprint arXiv:0912.0779*, 2009.

[150] Mohammad H. Amin. Searching for quantum speedup in quasistatic quantum annealers. *Physical Review A*, 92(5):1–6, 2015.

[151] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[152] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[153] David Ellerman. In quantum computing speedup illusory? The false coin of "counting function evaluations". *arXiv preprint arXiv:1407.4345*, 2014.

[154] Markus Hunziker, David A. Meyer, Jihun Park, James Pommersheim, and Mitch Rothstein. The geometry of quantum learning. *Quantum Information Processing*, 9(3):321–341, 2010.

[155] Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Hiroyuki Masuda, Raymond H Putra, and Shigeru Yamashita. Quantum identification of Boolean oracles. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*, pages 105–116. Springer, 2004.

[156] Alp Atici and Rocco A Servedio. Improved bounds on quantum learning algorithms. *Quantum Information Processing*, 4(5):355–386, 2005.

[157] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[158] Steve Hanneke. The optimal sample complexity of PAC learning. *Journal of Machine Learning Research*, 17(38):1–15, 2016.

[159] Nader H Bshouty and Jeffrey C Jackson. Learning DNF over the uniform distribution using a quantum example oracle. *SIAM Journal on Computing*, 28(3):1136–1153, 1998.

[160] Srinivasan Arunachalam and Ronald de Wolf. Optimal quantum sample complexity of learning algorithms. *arXiv preprint arXiv:1607.00932*, 2016.

[161] Andrew W Cross, Graeme Smith, and John A Smolin. Quantum learning robust against noise. *Physical Review A*, 92(1):012327, 2015.

[162] Jun-ichi Inoue. Application of the quantum spin glass theory to image restoration. *Physical Review E*, 63(4):046114, 2001.

[163] Jun-ichi Inoue. Pattern-recalling processes in quantum Hopfield networks far from saturation. In *Journal of Physics: Conference Series*, volume 297, page 012012. IOP Publishing, 2011.

[164] Masha Shcherbina and Brunello Tirozzi. Quantum Hopfield model. *arXiv:1201.5024v1*, 2012.

[165] Hidetoshi Nishimori and Yoshihiko Nonomura. Quantum effects in neural networks. *Journal of the Physical Society of Japan*, 65(12):3780–3796, 1996.

[166] Siddhartha Santra, Omar Shehab, and Radhakrishnan Balu. Exponential capacity of associative memories under quantum annealing recall. *arXiv preprint arXiv:1602.08149*, 2016.

[167] Alex Monras, Almut Beige, and Karoline Wiesner. Hidden quantum Markov models and non-adaptive read-out of many-body states. *Applied Mathematical and Computational Sciences*, 3, 2011.

[168] Alex Monras and Andreas Winter. Quantum learning of classical stochastic processes: The completely positive realization problem. *Journal of Mathematical Physics*, 57(1):015219, 2016.

[169] Jennifer Barry, Daniel T. Barry, and Scott Aaronson. Quantum partially observable Markov decision processes. *Physical Review A*, 90:032311, Sep 2014.

[170] Michał Cholewa, Piotr Gawron, Przemysław Głomb, and Dariusz Kurzyk. Quantum hidden Markov models based on transition operation matrices. *arXiv preprint arXiv:1503.08760*, 2015.

[171] Katja Ried, Megan Agnew, Lydia Vermeyden, Dominik Janzing, Robert W Spekkens, and Kevin J Resch. A quantum advantage for inferring causal structure. *Nature Physics*, 11(5):414–420, 2015.

[172] Časlav Brukner. Quantum causality. *Nature Physics*, 10(4), 2014.

[173] Fabio Costa and Sally Shrapnel. Quantum causal modelling. *arXiv preprint arXiv:1512.07106*, 2015.

[174] Cyril Stark. Learning optimal quantum models is NP-hard. *arXiv preprint arXiv:1510.02800*, 2015.

[175] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du. Experimental realization of a quantum support vector machine. *Physical Review Letters*, 114(14):140504, 2015.

[176] Xin-Dong Cai, Dian Wu, Zu-En Su, Ming-Cheng Chen, Xi-Lin Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Entanglement-based machine learning on a quantum computer. *Physical Review Letters*, 114(11):110504, 2015.

[177] Marisa Pons, Veronica Ahufinger, Christof Wunderlich, Anna Sanpera, Sibylle Braungardt, Aditi Sen, Ujjwal Sen, Maciej Lewenstein, et al. Trapped ion chain as a neural network: Error resistant quantum computation. *Physical Review Letters*, 98(2):023003, 2007.

[178] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2):022342, 2016.

[179] Iris Cong and Luming Duan. Quantum discriminant analysis for dimensionality reduction and classification. *New Journal of Physics*, 18:073011, 2016.

[180] Iordanis Kerenedis and Anupam Prakash. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675*, 2016.

[181] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum nearest-neighbor algorithms for machine learning. *Quantum Information & Computation*, 15:0318–0358, 2015.

[182] Giuseppe Davide Paparo and MA Martin-Delgado. Google in a quantum network. *Scientific Reports*, 2, 2012.

[183] T Loke, JW Tang, J Rodriguez, M Small, and JB Wang. Comparing classical and quantum pageranks. *Quantum Information Processing*, 16(1):25, 2017.

[184] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Quantum speed-up for unsupervised learning. *Machine Learning*, 90(2):261–287, 2013.

[185] Tamer Salman and Yoram Baram. Quantum set intersection and its application to associative memory. *Journal of Machine Learning Research*, 13(Nov):3177–3206, 2012.

[186] William Zeng and Bob Coecke. Quantum algorithms for compositional natural language processing. *arXiv preprint arXiv:1608.01406*, 2016.

[187] Dmytro Korenkevych, Yanbo Xue, Zhengbing Bian, Fabian Chudak, William G Macready, Jason Rolfe, and Evgeny Andriyash. Benchmarking quantum hardware for training of fully visible Boltzmann machines. *arXiv preprint arXiv:1611.04528*, 2016.

[188] Steven H Adachi and Maxwell P Henderson. Application of quantum annealing to training of deep neural networks. *arXiv preprint arXiv:1510.06356*, 2015.

[189] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum Boltzmann machine. *arXiv preprint arXiv:1601.02036*, 2016.

[190] Nathan Wiebe and Christopher Granade. Can small quantum systems learn? *arXiv preprint arXiv:1512.03145*, 2015.

[191] Rodion Neigovzen, Jorge L Neves, Rudolf Sollacher, and Steffen J Glaser. Quantum pattern recognition with liquid-state nuclear magnetic resonance. *Physical Review A*, 79(4):042321, 2009.

[192] Hadayat Seddiqi and Travis Humble. Adiabatic quantum optimization for associative memory recall. *Frontiers in Physics*, 2:79, 2014.

[193] Joseph Dulny III and Michael Kim. Developing quantum annealer driven data discovery. *arXiv preprint arXiv:1603.07980*, 2016.

[194] Kristen L Pudenz and Daniel A Lidar. Quantum adiabatic machine learning. *Quantum Information Processing*, 12(5):2027–2070, 2013.

[195] Issei Sato, Kenichi Kurihara, Shu Tanaka, Hiroshi Nakagawa, and Seiji Miyashita. Quantum annealing for variational Bayes inference. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 479–486. AUAI Press, 2009.

[196] Bryan OGorman, R Babbush, A Perdomo-Ortiz, Alan Aspuru-Guzik, and Vadim Smelyanskiy. Bayesian network structure learning using quantum annealing. *The European Physical Journal Special Topics*, 224(1):163–188, 2015.

[197] Vasil Denchev, Nan Ding, Hartmut Neven, and Svn Vishwanathan. Robust classification with adiabatic quantum optimization. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 863–870, 2012.

[198] Hideyuki Miyahara and Koji Tsumura. Relaxation of the EM algorithm via quantum annealing. *arXiv preprint arXiv:1606.01484*, 2016.

[199] David Horn and Assaf Gottlieb. Algorithm for data clustering in pattern recognition problems based on quantum mechanics. *Physical Review Letters*, 88(1):018702, 2002.

[200] Yiqian Cui, Junyou Shi, and Zili Wang. Lazy quantum clustering induced radial basis function networks (LQC-RBFN) with effective centers selection and radii determination. *Neurocomputing*, 175:797–807, 2016.

[201] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[202] Christoph Dürr and Peter Hø yer. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*, 1996.

[203] Julia Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.

[204] Hans J Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific Reports*, 2:1–16, 2012.

[205] Julia Kempe. Quantum random walks hit exponentially faster. *arXiv preprint quant-ph/0205083*, 2002.

[206] Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Physical Review A*, 94(2):022308, 2016.

[207] Marcello Benedetti, John Realpe-Gómez, Rupak Biswas, and Alejandro Perdomo-Ortiz. Quantum-assisted learning of graphical models with arbitrary pairwise connectivity. *arXiv preprint arXiv:1609.02542*, 2016.

[208] Krysta M. Svore Nathan Wiebe, Ashish Kapoor. Quantum deep learning, 2014. arXiv: 1412.3489v1.

[209] Nathan Wiebe, Ashish Kapoor, Christopher Granade, and Krysta M Svore. Quantum inspired training for Boltzmann machines. *arXiv preprint arXiv:1507.02642*, 2015.

[210] Hartmut Neven, Vasil S Denchev, Geordie Rose, and William G Macready. Qboost: Large scale classifier training with adiabatic quantum optimization. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, pages 333–348, 2012.

[211] Matthieu Courbariaux, Itay Hubara, COM Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to + 1 or -. *arXiv preprint arXiv:1602.02830*, 2016.

[212] Hartmut Neven, Geordie Rose, and William G Macready. Image recognition with an adiabatic quantum computer i. Mapping to quadratic unconstrained binary optimization. *arXiv preprint arXiv:0804.4457*, 2008.

[213] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2):022342, 2016.

[214] Carl Friedrich Gauss. *Theory of the Combination of Observations Least Subject to Error: Part One, Part Two, Supplement*, volume 11. Siam, 1995. Translated by G.W. Stewart from the original 1820 edition.

[215] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[216] Rupak Chatterjee and Ting Yu. Generalized coherent states, reproducing kernels, and quantum support vector machines. *arXiv preprint arXiv:1612.03713*, 2016.

[217] Patrick Rebentrost, Maria Schuld, Francesco Petruccione, and Seth Lloyd. Quantum gradient descent and Newton's method for constrained polynomial optimization. *arXiv preprint quant-ph/1612.01789*, 2016.

[218] Christodoulos A Floudas and Panos M Pardalos. *Encyclopedia of optimization*. Springer Science & Business Media, 2001.

[219] Simai He, Zhening Li, and Shuzhong Zhang. Approximation algorithms for homogeneous polynomial optimization with quadratic constraints. *Mathematical Programming*, 125(2):353–383, 2010.

[220] Yurii Nesterov et al. Random walk in a simplex and quadratic optimization over convex polytopes. Technical report, UCL, 2003. CIRE Discussion Papers 2003/71.

[221] Etienne De Klerk. The complexity of optimizing over a simplex, hypercube or sphere: A short survey. *Central European Journal of Operations Research*, 16(2):111–125, 2008.

[222] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *Journal of Machine Learning Research*, 48, 2016.

[223] Alan A Goldstein. Convex programming in Hilbert space. *Bulletin of the American Mathematical Society*, 70(5):709–710, 1964.

[224] Evgeny S Levitin and Boris T Polyak. Constrained minimization methods. *USSR Computational Mathematics and Mathematical Physics*, 6(5):1–50, 1966.

[225] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Quantum computing for pattern classification. In *Lecture Notes in Computer Science*, volume 8862, pages 208–220. Springer, 2014.

[226] Carlo A Trugenberger. Quantum pattern recognition. *Quantum Information Processing*, 1(6):471–493, 2002.

[227] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. How to simulate a perceptron using quantum circuits. *Physics Letters A*, 379:660–663, 2015.

[228] Heinz Peter Breuer and Francesco Petruccione. *The theory of open quantum systems*. Oxford University Press, 2002.

[229] Michael M Wolf. Quantum channels & operations: Guided tour. *Lecture notes available at http://www-m5. ma. tum. de/foswiki/pub M*, 5, 2012.

[230] Prabin Adhikari, Mohammad Hafezi, and JM Taylor. Nonlinear optics quantum computing with circuit QED. *Physical Review Letters*, 110(6):060503, 2013.

[231] Hayato Goto. Bifurcation-based adiabatic quantum computation with a nonlinear oscillator network. *Scientific Reports*, 6, 2016.

[232] Paul Pfeiffer, IL Egusquiza, M Di Ventra, M Sanz, and E Solano. Quantum memristors. *Scientific Reports*, 6, 2016.

[233] Ludmila I Kuncheva. *Combining pattern classifiers: Methods and algorithms*. John Wiley & Sons, 2004.

[234] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[235] Robert E Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[236] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the European Conference on Computational Learning Theory*, pages 23–37. Springer, 1995.

[237] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[238] Thomas P Minka. Bayesian model averaging is not model combination, 2000. Comment available electronically at http://www.stat.cmu.edu/minka/papers/bma. html.

[239] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[240] Louisa Lam and SY Suen. Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 27(5):553–568, 1997.

[241] Robert E Schapire. Explaining Adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[242] Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boixo, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy, and John Martinis. Commercialize quantum technologies in five years. *Nature*, 543(7644):171, 2017.

[243] Unai Alvarez-Rodriguez, Lucas Lamata, Pablo Escandell-Montero, José D Martín-Guerrero, and Enrique Solano. Quantum machine learning without measurements. *arXiv preprint arXiv:1612.05535*, 2016.