Raban Iten

Artificial Intelligence for Scientific Discoveries

Extracting Physical Concepts from Experimental Data Using Deep Learning

Springer

Artificial Intelligence for Scientific Discoveries

Raban Iten

Artificial Intelligence for Scientific Discoveries

Extracting Physical Concepts from Experimental Data Using Deep Learning

Springer

Raban Iten
ETH Zürich
Zürich, Switzerland

ISBN 978-3-031-27018-5 ISBN 978-3-031-27019-2 (eBook)
https://doi.org/10.1007/978-3-031-27019-2

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 Theres and Aschi Iten

Preface

This book is written for scientists interested in using machine learning to discover physical concepts. No prior knowledge of machine learning is required, but a first degree in a science subject is assumed.

The motivation to write this book is rooted in my Ph.D. research, which focused on the long-term goal of using machine learning to make progress on fundamental questions in modern physics. At the beginning of the Ph.D., I had only a very basic knowledge of machine learning, so I started reading a lot of papers in the field of machine learning as well as papers that used machine learning for physics. Most of the work using machine learning for physics focuses on finding model parameters for specific physical systems, and I could only find a few papers that use machine learning in a way to gain insights about the model itself rather than just its parameters. These papers come from a variety of research communities, and it took me some effort to get an overview of the relevant machine learning methods for automating parts of a physicist's discovery process. Therefore, I thought that providing an overview over this field could be helpful to other physicists who are interested in learning about how to use machines to get insights from experimental data. I would be very interested in any feedback on the book or suggestions for work to be covered in a second version. If you have any comments, please just email me at itenr@ethz.ch.

Zurich, Switzerland
January 2023

Raban Iten

Acknowledgements

This book is based on my Ph.D. and would not exist in this form without the close collaboration with Tony Metger and Hendrik P. Nautrup. I would like to warmly thank these brilliant friends for the fruitful and very enjoyable collaboration and for reviewing parts of this book.

In addition, I would like to thank Angela Lahee, Ashok Arumairaj and their team from Springer for their enthusiasm for the topic covered in this book and their great support while writing it.

I would also like to sincerely thank my Ph.D. supervisor Renato Renner for interesting discussions and for supporting me in writing this book. His precision in scientific thinking is highly inspiring and clearly sharpened my mind. Renato gave me a lot of freedom to work on topics of my interest and was always open to discuss any questions.

Furthermore, I would like to thank Giuseppe Carleo and Matthias Troyer for reviewing my work as co-examiners for my doctoral thesis.

I am also very grateful to Manfred Sigrist for interesting discussions and the occasional chats in the office or during a jogging round, and last but not least for creating a very friendly atmosphere on the K floor in the HIT building of ETH Zurich.

During the last couple of years, I was extremely lucky to collaborate with various brilliant people including Mario Berta, Hans J. Briegel, Matthias Christandl, Roger Colbeck, Emanuel Malvetti, Hendrik P. Nautrup, Tony Metger, Romain Moyard, Navneeth Ramakrishnan, Joseph M. Renes, Renato Renner, Henrik Wilming and Stefan Woerner. I would like to thank all of them for the interesting collaborations.

Hans J. Briegel, Mario Krenn, Alexey Melnikov, Tony Metger, Hendrik P. Nautrup, Henrik Wilming and Monika Zimmermann helped me to improve this book, and I thank them very much. I would also like to thank Mario Krenn for very interesting and extremely motivating discussions about how to use machine learning to discover conceptual information.

In addition, I would like to thank all the former and current members of the quantum information theory group at ETH Zurich for many interesting and stimulating discussions. In particular, I would like to thank my office mates Rotem Arnon-Friedman and Mischa Woods for many interesting discussions and their scientific advice. Further, I would like to thank Christa Zoufal for helping me to organize an internship at IBM Research in Zürich with Stefan Woerner during my Ph.D., which was a very enriching experience and a truly wonderful time, not only because of the delicious food at the IBM canteen.

Furthermore, I have to thank my girlfriends along the Ph.D. for setting me challenges outside of the physics world and thus contributing to my personal learning experience. They taught me complex lessons (sometimes painful to learn) and provided me with various experiences, each of them in their unique way. This contributed greatly to making my Ph.D. time an instructive period of my life. In particular, I would like to thank Monika for creating a steady, supportive and stimulating social environment which led to a productive time towards the end of my Ph.D.

Finally and most importantly, I would like to thank my two sisters and my parents for their constant support and encouragement.

Zurich, Switzerland Raban Iten
January 2023

Contents

1 Introduction . . . 1
- 1.1 Motivation . . . 2
- 1.2 Physicist's Discovery Process . . . 3
- 1.3 Extracting Relevant Parameters from Experimental Data . . . 4
- 1.4 Outline . . . 6

Part I Machine Learning Background

2 Machine Learning in a Nutshell . . . 11
- 2.1 Supervised Learning . . . 11
- 2.2 Unsupervised Learning . . . 12
- 2.3 Reinforcement Learning . . . 13
- 2.4 Bias-Variance Tradeoff . . . 14

3 Artificial Neural Networks . . . 17
- 3.1 Single Artificial Neuron . . . 17
- 3.2 Activation Functions . . . 18
- 3.3 Neural Networks . . . 19
- 3.4 Universality Theorem . . . 20
- 3.5 Training of Neural Networks . . . 21
  - 3.5.1 Stochastic Gradient Descent . . . 22
  - 3.5.2 Convergence and Choice of Hyperparameters . . . 23
  - 3.5.3 Generalization . . . 24
- 3.6 Deep Learning . . . 25

4 Autoencoders . . . 27

Part II Overview of Using Machine Learning for Scientific Discoveries

5 Creating Experimental Setups . . . 33
- 5.1 Problem Setting from Quantum Optics . . . 34
  - 5.1.1 Entanglement of Bipartite Systems . . . 34
  - 5.1.2 Entanglement of Multipartite Systems . . . 36

5.1.3 Preparation of Photon States 36
5.2 Creating Experimental Setups Using Projective Simulation 37
5.2.1 Architecture for PS Agent 39
5.2.2 Training of the PS Agent 40
5.2.3 Results 44
5.3 Conceptual Insights from Action Composition 45

6 Model Creation 47
6.1 Optimizing Model Parameters 47
6.2 Discovering Physical Laws 48
6.2.1 Symbolic Regression 49
6.2.2 Extracting Physical Laws from Data 52

7 Model Testing 57
7.1 Statistical Setting 58
7.2 Approximation of Probability Density Functions 60
7.2.1 Parametric Modelling 61
7.2.2 Non-parametric Modelling 62
7.3 Statistical Hypothesis Testing 63
7.4 Identifying the Discrepant Regions 64
7.5 Application for Model Testing 65

Part III Representation Learning for Physical Discoveries

8 Theory: Formalizing the Process of Human Model Building 69
8.1 Motivation 69
8.2 Physicist's Reasoning Process 70
8.3 Experimental Setting and Data Creation 70
8.4 Criteria for Operationally Meaningful Representations 72
8.4.1 Minimal Representation 72
8.4.2 Separation of Physical Parameters 73
8.4.3 Simple Update Rules 81
8.5 Criteria for Mathematically Meaningful Representations 82
8.5.1 Koopman Operator Theory 83
8.5.2 Representation of Koopman Eigenfunctions 84
8.6 Criteria for Physically Meaningful Representations 85
8.6.1 Interaction Graph 86
8.6.2 Representation of Interaction Graph 86

9 Methods: Using Neural Networks to Find Simple Representations . 89
9.1 Motivation 89
9.2 General Network Structure to Learn Representations 90
9.3 Network Structures for Separating Parameters 92
9.3.1 Statistically Independent 92
9.3.2 Operationally Meaningful 93
9.4 Network Structure to Find Representations with Simple Update Rules 97

9.5 Optimality Guarantees on the Representation 99
9.6 Network Structure to Find Koopman Eigenfunctions 99
9.7 Network Structure to Find Interaction Graphs 100
9.7.1 Graph Neural Networks 101
9.7.2 Network Structure to Learn Interaction Graphs 102

10 Applications: Physical Toy Examples 107
10.1 Motivation 108
10.2 Damped Pendulum 109
10.3 Dynamics of the Nonlinear Pendulum 111
10.4 Conservation of Angular Momentum 112
10.5 Representation of Qubits 114
10.5.1 Minimal Representations for Pure Quantum States 116
10.5.2 Local Representation of Two-Qubit States 119
10.6 Charged Particles 121
10.6.1 Single Encoder 123
10.6.2 Multiple Encoders 125
10.7 Heliocentric Solar System 126
10.8 Several Particles Connected by Springs 128

11 Future Research Directions and Further Reading 131
11.1 Finding Measurement Strategies and Representations 131
11.2 Interpretability and Generalization of *SciNet* 133

Part IV Future Prospects

12 Future Prospects 137
12.1 AI Physicist 137
12.2 Learning Procedures Instead of Simple Functions 139
12.3 AI for Foundations of Physics 141

Appendix A: Interpretation of the Number of Latent Variables 143

Appendix B: Variational Autoencoders 147

Appendix C: Implementation Details 151

Appendix D: Representations of Cyclic Parameters 155

Appendix E: Classical Mechanics Derivation for Charged Masse 157

References 161

Chapter 1
Introduction

There has been amazing progress in the field of artificial intelligence (AI) and machine learning in recent years. Not long ago, robots could not walk, now they can even dance (as for example demonstrated by robots from Boston Dynamics in 2020). Computer vision and natural language processing have reached a level of performance that make them indispensable in many areas today, such as face recognition, speech interpretation and self-driving cars. While most AI is developed based on domain knowledge, i.e., specific knowledge about the data under consideration or the environment of interest, there are also impressive results where AI learns from scratch. A breakthrough in this direction was achieved with MuZero by DeepMind [1]. MuZero is a machine learning system that learns to play Go, chess and shogi from scratch without knowing anything about the rules of the games or what it would take to win. You may wonder how it is possible to learn a game without watching a human play the game and learning from him or her. The strategy used by DeepMind is to let machines play against each other and tell them when they have won a game. Surprisingly, this information (together with the information about which moves are allowed) is sufficient for the machine to build a model of the game and its rules and develop a winning strategy without learning from a human player. Impressively, the trained machine has been able to beat the best human players in the world in all three games, Go, chess and shogi. Having algorithms that can learn models of different games without any prior knowledge about the games, nothing prevents us from applying such algorithms to scientific problems. Indeed, the methods used for MuZero have already been successfully applied to a range of complex problems in chemistry [2] and quantum physics [3]. However, in science, we are not only interested in the prediction, but also in how we make the prediction. The main goal of physicists is to find the simplest models that describe nature. Having such a model at hand makes it possible to predict the behavior of previously unobserved systems. Unfortunately, understanding the model learned by MuZero remains very challenging and hence we do not get much insight into how MuZero makes its predictions. This is a fundamental problem often faced when applying machine learning

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_1

to scientific problems. As scientists, we are particularly interested in understanding *how* the machine learning system makes its predictions and hence what the underlying model looks like. In this book we focus on this question, i.e., on how we can extract conceptual information from physical systems using machine learning. This question is of interest for several reasons. On one hand, it is an important step for the long term goal of building AI physicists [4], on the other hand, it may help us to gain insight into fundamental problems in modern physics. The latter is explained in more detail in the following section.

1.1 Motivation

History has shown that physical theories often only provide accurate predictions under certain restrictions of the environment under consideration. For example, Newtonian physics works extremely well for describing the behavior of objects we know from our daily life like a table or a football. However, the theory does not apply to extremely small objects like molecules. Rather, physical phenomena on the length scale of atoms are described more precisely by quantum theory. New physical theories usually contain certain elements of previously known theories. This is expected since the new theory should be more fundamental than the previous one and hence, we expect to find the previous theory as a limit case of the new theory. In the mentioned case, we recover Newtonian physics from quantum physics by considering the limit of vanishing Planck constant. However, transferring too many elements from the previous theory to a new theory may not lead to the simplest model to explain experimental data, but rather to the one that most naturally followed from the previous theory.

For example, before the discovery of special relativity, the constant speed of light that appears in Maxwell's equations of electromagnetism was explained by a medium called "aether" in which light can propagate. The aether served as an absolute and unique frame in which Maxwell's equations hold. The idea of aether goes back to the seventeenth century. Robert Boyle considered two sort of aethers, one serving as a medium that explains mechanical interactions between separated bodies, and another one explaining phenomenas like magnetism [5]. Therefore, the idea of the aether was already strongly based on prior knowledge of physics at the time of its introduction, namely the idea that objects can only interact with each other through other particles. Although the idea of the aether became more magical over time and the theory came into conflict with several experiments, the idea survived until the twentieth century, when Einstein came up with the special theory of relativity. The essential idea of special relativity is that light propagates with the same speed with respect to every inertial system. A conclusion of this statement is that time and space cannot be considered separately: two events that occur at the same time for one observer, may occur at different times for another observer. Special relativity required a complete rethink of our physical intuition and we had to abandon the idea of absolute time and an aether as a medium for the propagation of light. This historical example shows that prior knowledge of physics can prevent us from finding the simplest explanations to describe observational data.

Nowadays, it could be that we are in a similar situation: although we can use quantum theory—which is built on classical mechanics—to predict measurement outcomes with very high accuracy in the realm of the very small, we still face several conceptual problems. One such problem, related to the measurement problem of quantum physics, is the question if an observer of a quantum system can itself be described by quantum mechanics. This problem was raised by Wigner in [6], deepened in [7] and recently sharpened in [8, 9]. Experimentally, it is investigated whether we can imitate observers with autonomous agents [10, 11]. A serious problem in modern physics, which may be related to the conceptual problems encountered in quantum physics, is that general relativity—impressively successful in the restricted regime of the very large—is fundamentally incompatible with quantum mechanics, as evidenced by paradoxes such as the black hole information loss [12, 13]. This raises an interesting question: are the laws of quantum physics (or other physical theories) the most natural ones to explain data from experiments if we assume no prior knowledge of physics? While this question is unlikely to be answered in the near future, the recent advances described in this book make a first step in this direction.

1.2 Physicist's Discovery Process

Building machines that can autonomously learn physical principles by interacting with an environment is an extremely challenging task. Roughly, the cyclic process a physicist goes through to find laws in nature can be divided into the following parts.

- **Choose a physical subsystem**: As humans, we collect a lot of data with our sensory organs every second. A physicist focuses on a particular part of this data. Usually, the data on which is focused is related to one or several physical subsystems, such as the data of a pendulum together with the air in its environment.
- **Create an experimental setup**: Create devices that are able to collect relevant data from physical processes.
- **Observations**: The physicist collects data about the chosen physical systems by using devices of an experimental setup.
- **Create a model**: Analyze observed data to build a (or expand an already existing) model describing the behavior of the underlying physical system. A useful model should make predictions that are experimentally falsifiable.
- **Test the model**: Test predictions made by a model. If the model is falsified, the physicist should create another model (or at least analyze the restricted scope of validity of the model). If it is not falsified, the model can be further tested and used for applications.

Each of these parts can be very challenging, and there are different approaches from machine learning trying to automize them. We provide an overview over such approaches in Part II. In Part III of this book we focus on one particular step of the model building process, namely the determination of the relevant degrees of freedom of a physical system from experimental data. Recent advances in this field have made it possible to discover compact representations of physical systems without requiring any prior knowledge of the considered physical system other than the given experimental data. Minimizing the prior knowledge is not only interesting for the long-term goal of finding an alternative description of quantum mechanics, but also an essential requirement for building an AI physicist [4]. Indeed, otherwise it could not be claimed that the AI physicist discovered the found laws completely by itself. The basic idea of the approach presented in Part III is explained in Sect. 1.3.

1.3 Extracting Relevant Parameters from Experimental Data

The goal of using machines to discover physical laws underlying experimental data has been pursued in several contexts (see Part II for a detailed discussion and [14–17] for recent reviews). A lot of early work focused on finding mathematical expressions describing a given dataset (see e.g. [18–20] and Sect. 6.2 for a detailed discussion of these works). For example, in [19] an algorithm recovers the laws of motion of mechanical systems, like a double pendulum, by searching over a space of mathematical expressions on given input variables (searching the form as well as the parameters of mathematical expressions is known under the term *symbolic regression* [21]). However, such approaches require prior knowledge in the form of knowing what the relevant variables are. In certain situations one might not have such prior knowledge or does not want to impose it to allow the machine to find entirely different representations of the physical system. Defining properties of such representations and describing how they can be found using machine learning is the main focus of this book (Part III) and is regarded as a first step towards discovering physical concepts with machine learning without using prior knowledge about physics and maths. In fact, in order to write down a physical law, we must first know which parameters occur in the law and thus which degrees of freedom the physical system has. In the following, the basic idea for finding the relevant degrees of freedom is presented.

Finding a minimal set of parameters that is sufficient to describe a given data set (without using any prior knowledge about the system that created the data set) is a complex task, but essential for the process of building physical models. Let us consider a very simple toy example, where we are given time series data $(t_i, x(t_i))_{i\in\{1,\ldots,N\}}$ of a particle moving along the x-direction with constant speed (for some $3 \leq N \in \mathbb{N}$). The goal of a physicist could be to build a model that allows to predict the x-coordinate of the position of the particle for any time t'. The standard approach is to extract the speed $v = (x(t_1) - x(t_0))/(t_1 - t_0)$ and the start position $x_0 = x(t_0) - vt_0$ from the data and then calculate $x(t') = x_0 + vt'$. Hence, the models that a physicist builds do not deal with the observations directly, but rather with a representation of the underlying physical state of the observed system, e.g., the two parameters *initial*

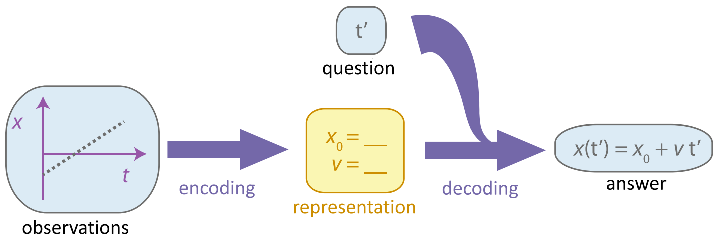


Fig. 1.1 Human learning of physical representations. (Figure reproduced from Iten and Metger et al., *Physical Review Letters*, 2020 [22]). A physicist compresses experimental observations into a simple representation (*encoding*). When later asked any question about the physical setting, the physicist should be able to produce a correct answer using only the representation and not the original data. We call the process of producing the answer from the representation *decoding*. For example, the observations may be the first few seconds of the trajectory of a particle moving with constant speed; the representation could be the parameters "speed v" and "initial position x_0" and the question could be "where will the particle be at a later time t'?"

position and *speed*, (x_0, v). Which parameters are used is an important part of the model. In Part III of this book we follow the ideas presented in [22, 23] and formalize what we understand by a "natural" representation and describe a machine learning system that finds such representations without using any prior knowledge about the specific physical system that is considered. One natural criteria for the representation is that it is sufficient, hence that it contains all relevant information to predict the system's (future) behavior. To know which parameters are required for a sufficient representation, we also have to model the prediction process of a physicist.

We hence split the human modeling process into two interdependent tasks. Firstly, finding the relevant parameters in experimental data (e.g., the *initial position* and *speed*, (x_0, v)). And, secondly, building a physical model that specifies how to make predictions (i.e., answer questions) based on the knowledge of the relevant parameters of the system (e.g. "where is the particle at time t'?"). Since our goal is to build a machine that solves these tasks, it is helpful to formalize them. Formally, the physical modelling process can be regarded as an "encoder" $E : \mathcal{O} \to \mathcal{R}$ mapping the set of possible observations $\mathcal{O}$ to representations $\mathcal{R}$, followed by a "decoder" $D : \mathcal{R} \times \mathcal{Q} \to \mathcal{A}$ mapping the sets of all possible representations $\mathcal{R}$ and questions $\mathcal{Q}$ to answers $\mathcal{A}$ (Fig. 1.1). Throughout this book, we refer to this structure as *SciNet* for simplicity. From a machine learning perspective, the structure of *SciNet* is closely related to autoencoders, a method from representation learning for finding compressed representations of high dimensional data (see Chap. 4 for a non-technical introduction to autoencoders). We would like to allow for arbitrary encoder and decoder functions, and let a computer learn these by itself from a given data set containing triples of the form $[o, q, a^\star(o, q)]$, where $a^\star(o, q)$ describes the correct answer to question q given observation o. Similar to [19], one could try to approach this task by using symbolic regression (discussed in Sect. 6.2.1) to find functions from input-output pairs for the encoder and the decoder synchronously. Thereby, one uses a certain kind of search algorithm based on evolutionary computation [24, 25], to search through the space of mathematical expressions fitting the input-output pairs. However, such an approach faces several challenges:

1. The outputs of the encoder are not given. Hence, one has to fix a mathematical expression describing the encoder (note that different expressions are allowed to have different output-dimensions) and then search trough the space of mathematical expressions representing the decoder to see if there exists a suitable one that provides the correct answers. Such a search is computationally extremely costly.
2. For common physical problems, the decoder function that calculates the predictions from the (small number of) relevant physical parameters is usually quite simple (in the example considered above, it has to implement the function $x(t') = x_0 + vt'$ with input (x_0, v, t')). However, the encoder function might be of a more complex form, and hence difficult to find with symbolic regression.
3. The input to the encoder can be very high dimensional, and a lot of the input parameters might not be relevant for answering the questions. Finding the relevant inputs by looping trough all possible subsets of input parameters is computationally extremely costly.
4. The search over functions is biased towards simple mathematical expressions. However, such a bias might not be desirable.

Artificial neural networks, introduced in Chap. 3, on the other hand can approximate any (continuous) function (Sect. 3.4), are unbiased towards simple mathematical expressions, perform very well in selecting only the relevant features from input data, and, probably most importantly, implementing the encoder and decoder with neural networks allows to learn both functions simultaneously. Crucially, having no restrictions on the encoder and decoder functions E and D, the proposed scheme does not use any prior knowledge about a specific physical system or mathematics. These are the main reasons why the methods discussed in Part III for finding the relevant parameters of physical systems are based on neural networks. A difficulty that arises due to approximating functions with neural networks is that it can be difficult for humans to interpret the found encoding mapping. We discuss some methods to overcome this difficulty, but a general solution is not yet known and is being actively researched in the field of machine learning.

1.4 Outline

The book is structured as follows. First, we provide a brief introduction into machine learning in Part I, which contains the computer science background for a scientist that is necessary to understand the following parts of the book. Readers who are familiar with the concepts of machine learning and artificial neural networks can safely skip this part.

The following two parts, Parts II and III, are self-contained and the reader is encouraged to directly focus on the part he is mostly interested in.

In Part II, we describe a selection of work that uses machine learning for automatizing different steps of a physicist's discovery process. Often, machine learning is applied for specific physical systems and prior knowledge about the system at hand is built into the machine learning structure. In contrast, the chapters in Part II focus on work that is applicable to a broad range of systems and uses a minimal amount of prior knowledge. The chapters in Part II are self-contained and the reader may pick the ones he or she is particularly interested in. In Chap. 5, we discuss how experimental setups can be created using reinforcement learning. In Chap. 6, we discuss model creation and optimization under the assumption that the relevant parameters of the considered systems are known. And finally, in Chap. 7, we investigate how to test a model that makes probabilistic predictions and how to find clusters in the experimental data that deviates the most from the model predictions.

Part III focuses on a particular part of model creation, namely extracting the relevant degrees of freedom from experimental data using minimal prior knowledge about physics or maths. The main idea to achieve this is to map a simplified version of the physicist's reasoning process (Fig. 1.1) to an artificial neural network structure. First, in Chap. 8, we formalize the setting and provide a mathematical formulation of the simplified physical reasoning process as well as of the properties of "natural" representations. In Chap. 9, we then describe machine learning methods that allow to find such representations. In Chap. 10, we apply these methods to various toy examples from classical and quantum physics, demonstrate that they allow to find the same variables that are used by physics textbooks and explain how conceptual insights can be extracted from the found representations. Finally, we conclude with some open questions and directions for future work on representation learning for physics in Chap. 11.

In the last part of the book (Part IV), we discuss future perspectives for automating the discovery process of physicists and the use of machine learning for the foundation of physics.

Part I
Machine Learning Background

In this part, we give a rough overview of machine learning whose aim is to create software that can learn by itself from given data or by interacting with an environment. We discuss the three important subfields of *supervised*, *unsupervised* and *reinforcement* learning. While the first two subfields consider systems that learn from given data sets, in reinforcement learning an agent interacts with an environment and hence, the input data is created continuously and may depend on the actions chosen by the artificial agent. While the data for supervised learning has to be labeled with the expected outputs, in unsupervised learning no such labels are provided.

After discussing the basics of machine learning, we introduce the reader to artificial neural networks, the (currently) most important tool for machine learning and essential for Part III of this book. Roughly, neural networks can represent and implement functions on real input data. We discuss the basic structure of neural networks, which kind of functions they can approximate and how to train them to learn a function from input–output pairs.

Finally, we introduce the reader to *autoencoders*, a tool for unsupervised learning or more concretely for *representation learning*. Representation learning aims to find "natural" and low-dimensional representations of high-dimensional data and builds the basis for Part III of this book.

Chapter 2
Machine Learning in a Nutshell

Machine learning (ML) has started to gain traction over the past years and found a lot of applications in science and industry. The main idea is to create algorithms that can learn from data themselves. Traditionally, we can divide ML into *supervised*, *unsupervised* and *reinforcement* learning. The idea behind machine learning is simplest explained with supervised learning, which is covered in the following section.

2.1 Supervised Learning

One celebrated application of ML is the recognition of objects on images. Let us assume that you are asked to write a program that takes images of cats and dogs as an input, and the program should output which of these animals is shown on the image. Writing an algorithm for this task would be extremely complex, since there are all sorts of cats and dogs, they might not be fully visible on the image, the image might be of bad quality and so on. Instead we could let us inspire from the human learning process. How do we train a child to distinguish cats and dogs? We show him or her a lot of examples of cats and dogs and the child will learn to distinguish them by him- or herself. But it would be pretty impossible to teach the child a recipe (which could be translated into an algorithm) of how to distinguish cats and dogs. This leads us to the main idea behind ML: we provide a lot of examples of labeled images (where we have the two labels "cat" and "dog"), the so called *training data*, to a computer algorithm, whose task is to output a (probably extremely complex) function F that takes an image as an input and provides the correct label as an output (see Fig. 2.1). The difficult task of writing directly a program implementing the function F is now shifted to designing an algorithm that is able to learn a good approximation of the function F from labeled images. This is exactly the goal of *supervised* machine learning. Clearly, this is a very difficult task, since the algorithm has to create a function that recognizes the relevant features that distinguish cats from dogs from a set of images. Otherwise, if the algorithm just created a function that memorizes the correct labels for the provided images in the training set, it could

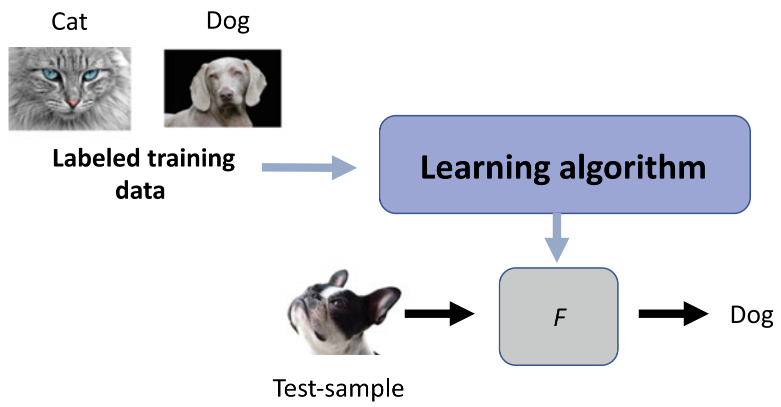


Fig. 2.1 Image classification. An common example for supervised machine learning is image classification. A learning algorithm is provided with a data set of labeled images (the so called *training data set*). For example, this could be a set of images of cats and dogs together with the correct label "cat" or "dog". The task of the learning algorithm is to infer a function F (which might be given as a blackbox), which is able to correctly label images. If the learning algorithm performs well, the inferred function F should also generalize, i.e., it should also label images correctly that were not seen during the training

not generalize to images not seen during the training. One option to create such an algorithm is to use artificial neural networks. For the present purpose, one may just think of a neural network as a parametrized family of functions $\{F_\theta\}_\theta$ from the space of images to the output labels (see Chap. 3 for the details). A training algorithm is then used to optimize over the choice of the parameters θ (determinig the function F_θ) using the labeled images with the goal that $F_{\theta^\star}$ gets close to the function F for some parameters $\theta^\star$. Interestingly, neural networks perform very well for this task and can generalize to images never seen during the training and without hardcoding specific information about cats and dogs into the network structure or the learning algorithm (see for example [26]).

2.2 Unsupervised Learning

Whereas in supervised learning one is given data samples together with the desired output for all the given samples, in *unsupervised* learning, one tries to learn some features about the given data without knowing the correct outputs for the training data. For example, given again images of cats and dogs, one might want to compress the images, i.e., to reduce the dimensionality of the original image, still maintaining all essential information about the image. Here, we may not know how such a representation should look like and we would like the algorithm to find the optimal one. The probably most widely known method for dimensionality reduction is *Principal Component Analysis* (PCA). Again, the goal of PCA is to reduce the dimension of

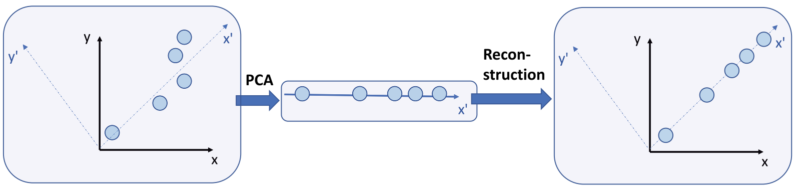


Fig. 2.2 Principal Component Analysis (PCA). A well known method for unsupervised learning is PCA, which exploits linear correlations in a data set to lower its dimension without loosing much information. The figure shows some data points in two dimensions (x, y) with a positive correlation of the x- and the y-coordinates. PCA transforms the coordinate system to (x', y'), such that the data points approximately lie on the x'-axis. In other words, the distribution of the y'-coordinate is approximately independent of the x'-coordinate. The y'-coordinate can be ignored without loosing too much information. Hence, PCA (approximately) reduced a two-dimensional data set to a one-dimensional one. The original distribution of the data can then be approximately recovered from the one-dimensional representation

the input data, but one restricts the allowed transformations from the input data to the new representation to be linear (Fig. 2.2). Finding compressed representations of data is the focus of *representation learning*, a subfield of unsupervised learning (see [27] for a recent review).

2.3 Reinforcement Learning

In *reinforcement learning*, the training data is not given, but is collected by an artificial agent from an environment. Hence, reinforcement learning is closely related to the objective of artificial intelligence (AI) of creating an agent that can perform actions in an environment to achieve a certain goal. The "environment" is considered to be everything of interest that is not part of the agent itself. For example, the environment of an agent playing chess against another agent could be the chess board and the other agent. The basic model for reinforcement learning is shown in Fig. 2.3. At a (time) step t, an agent gets the state $s^{(t)}$ of an environment (encoding some information of the environment like positions of objects) and a reward $r^{(t)}$ quantifying its performance in achieving the desired goal as an input. Based on the input, the agent chooses an action $a^{(t)}$ of a set of possible actions. The action transforms the environment from a state $s^{(t)}$ to a state $s^{(t+1)}$. In the example of plying chess, the action could for example be the movement of a game figure, and the new state $s^{(t+1)}$ could correspond to an encoding of all the positions of the game figures after the movement. Further a reward $r^{(t+1)}$ is returned to the agent, providing information about how useful the action $a^{(t)}$ was in getting closer to achieving the agent's goal. Typical examples for reinforcement learning environments are games like chess or Go, where the goal of the agent is simply to win the game. The actions of the agent are restricted to the ones allowed by the rules of the game and the reward could be zero as long as the game is not finished, and -1 or 1 if the agent looses or wins the game, respectively.

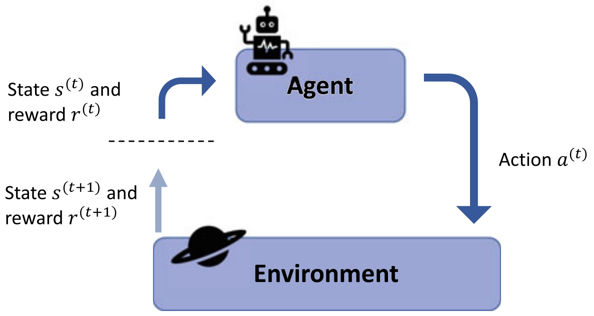

State $s^{(t)}$ and reward $r^{(t)}$
Agent
Action $a^{(t)}$
State $s^{(t+1)}$ and reward $r^{(t+1)}$
Environment

Fig. 2.3 Basic setting for reinforcement learning. An agent has to achieve a given goal in an environment. In the t-th interaction step, the agent gets an encoding $s^{(t)}$ of the current state of the environment and a reward $r^{(t)}$ quantifying its current performance as an input. Based on this input, the agent decides which action $a^{(t)}$ to perform next. The action transforms the environment from a state $s^{(t)}$ to a state $s^{(t+1)}$, and returns the reward $r^{(t+1)}$. This cycle is then repeated until the agent achieves its goal

2.4 Bias-Variance Tradeoff

The main goal of machine learning is to learn a model from a training data set (or a training environment) that generalizes well to new data samples (or new environments) not seen during training. The difficulty thereby is to handle the tradeoff between two kind of errors, the so called *bias* and *variance* of a machine learning algorithm. Roughly, bias is the tendency of an algorithm to consistently learn the same wrong thing, where variance is the tendency to learn random things not related to the real data.

Let us explain this with a simple example that should be familiar to most readers: fitting a polynomial trough 5 given data points (x_i, y_i) with $i \in \{1, \ldots, 5\}$ (see Fig. 2.4). The points are generated by a linear model with some Gaussian noise, i.e., $y_i = cx_i + \varepsilon z_i$ for constants $c \in \mathbb{R}$, $\varepsilon > 0$ and $z_i \sim \mathcal{N}(0, 1)$ sampled from the normal distribution. Assume now that we do not know exactly the underlying model for the data creation, but our guess is that it is a polynomial model (with some noise). One may for example try a linear fit to the data and one with a polynomial of degree four. A linear model is more restrictive and is described with less parameters (just a constant shift in the y-direction and the slope of the line). From a machine learning perspective, we expect such a model to lead to a larger error due to the low expressivity of the model, i.e., a such a model has a higher bias than a polynomial of degree four (for a mathematical discussion of the bias and variance we refer to [28]). In the example demonstrated in Fig. 2.4, we find that the linear model can not predict the data point perfectly, but actually quite well for such a simple model. On the other hand, fitting a polynomial of degree four clearly can fit the data point perfectly and has no error due to the simplicity of the model. Unfortunately, this comes in tradeoff

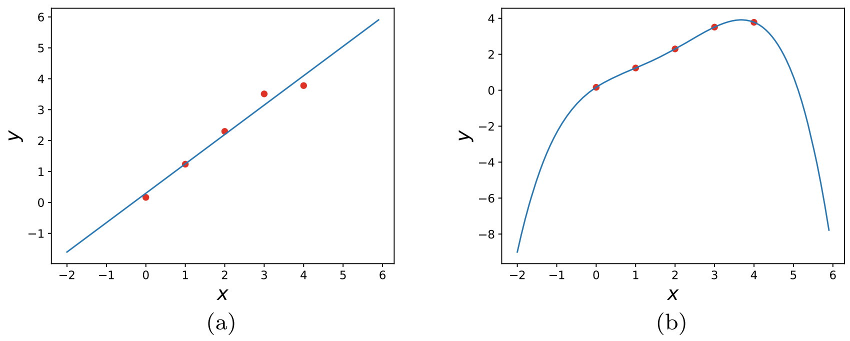


Fig. 2.4 Bias-variance tradeoff for polynomial regression. The data points are given by (x_i, y_i) with $x_i = i - 1$ and $y_i = x_i + 0.5z_i$ with randomly sampled $z_i \sim \mathcal{N}(0, 1)$ and $i \in \{1, \ldots, 5\}$. **a** Linear least squares fit of the data. **b** Polynomial least squares fit of the data with a polynomial of degree four. Although this polynomial perfectly fits the data points, it does not reflect the underlying linear model. Sampling a new data point (x, y) with $x = 6$, the prediction of this model would be around -8, which is far away of the expected value of $y = 6$. This demonstrated the tradeoff between fitting the training data and generalizing well to the test data. The tradeoff between making prediction errors due to too simplistic models and overfitting the training set with too expressive models is called the *bias-variance tradeoff* in machine learning

with a higher variance, meaning that the fit depends strongly on the samples in the training set, i.e., in particular on the random samples $z_i \sim \mathcal{N}(0, 1)$. The problem with models that are too expressiv do not directly manifest itself on the training data (we have zero prediction error there!), but will lead to bad generalization behavior. Indeed, in the example in Fig. 2.4, predicting the value of y for a test sample with $x = 6$ would be close to $y = -8$, where the expected value is $y = 6$. We say that we *overfitted* the training data by choosing a model with a too high variance.

In the example above we knew the underlying model, and hence, it was clear that the linear fit will have better generalization power. However, in general, the underlying model is not known. It could be that the data samples are indeed created by a four dimensional polynomial with zero noise. In this case, choosing a model with high bias such as a liner fit, would fit and generalize badly. This demonstrates a fundamental problem of machine learning, the so called *bias-variance tradeoff* stating that we can not lower both errors, the variance and the bias, at the same time. The goal of most machine learning strategies is to find the optimal tradeoff of the variance and the bias to achieve good generalization properties.

Remark 2.1 (Cross-validation) In the example discussed above, it remains unclear how to systematically test the generalization power of a machine learning model. One option is *cross-validation*, which partitions the training set into two complementary subsets and trains the model on one of the subsets. The other subset is then used to test the generalization power of the trained model. This is done with different partitions of the training data. If the model performs well on the training subsets, but badly generalizes to the complementary subset, this points towards the fact that we have a high variance and hence that our model has too high expressivity (or is not regularized properly). On the other hand, if our model has high prediction error on the training subsets as well as on the complementary subsets, this points towards a model with too high bias, and hence we may want to replace the model with one with higher expressivity.

Chapter 3
Artificial Neural Networks

Artificial neural networks have become the state of the art for tackling machine learning problems and to build AI-agents. Furthermore, they are considered for gaining insights into how the human brain develops physical intuition from observations [29–35]. For example, in [36] neural network were shown to be able to predict whether a tower consisting of wooden blocks is stable or will fall. Such experiments suggest that the processing of neural networks may have similarities to the intuitive reasoning of humans. In this section, we provide a short overview of the basics of neural networks. We still do not have a deep understanding of the training process of such networks and why they often generalize well to data not seen during the training. Hence, training neural networks requires some experience and knowledge about several tricks, and can not be taught easily. The book by Michael Nielsen on neural networks and deep learning [37] provides an overview of several such tricks. Understanding how to train neural networks is a subject of current research. For example, an information theoretic approach was suggested in [38]. Here we focus on a basic understanding of what neural networks are and how the training method works in principle.

3.1 Single Artificial Neuron

The building blocks of neural networks are single neurons (Fig. 3.1a). We can think of a neuron as a map that takes several real inputs $x_1, \ldots, x_n$ and provides an output $\sigma(\sum_i w_i x_i + b)$, according to an *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$, where the weights $w_i \in \mathbb{R}$ and the bias $b \in \mathbb{R}$ are tunable parameters. The output of the neuron is itself sometimes denoted by *activation*, and there are different possible choices for the activation function.

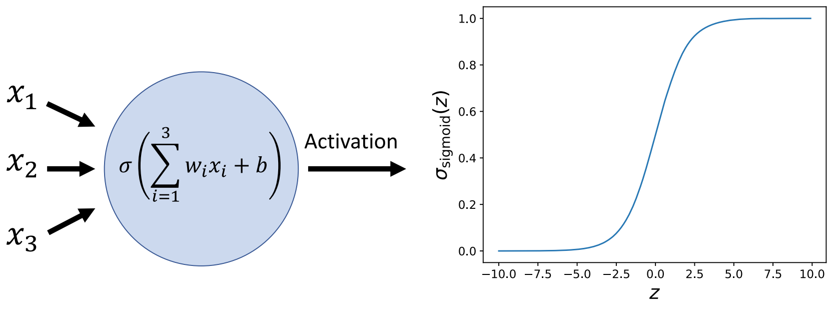

x_1
x_2
x_3
$\sigma\left(\sum_{i=1}^{3} w_i x_i + b\right)$
Activation
$\sigma_{\text{sigmoid}}(z)$
z
(a)
(b)

Fig. 3.1 Sigmoid neuron. **a** Single artificial neuron with weights w_i, bias b and activation function σ. The inputs to the neuron are denoted by x_1, x_2 and x_3. The output of the neuron is called activation and is given by $\sigma(\sum_i w_i x_i + b) \in \mathbb{R}$. **b** Sigmoid activation function given in (3.2). The sigmoid activation function can be seen as a smooth approximation of the step function given in (3.1). Hence, the neuron provides an output that is approximately equal to one if the input is significantly bigger than zero, and it provides an output close to zero if the input is significantly smaller than zero

For example, consider a step function

$$\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} . \tag{3.1}$$

Such a neuron is called perceptron and was developed in the 1950s and 1960s by Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. The choice of such an activation function is (loosely) motivated by the functioning of a biological neuron, which only "fires" (i.e., produces an output) if the sum of the weighted input signals is higher than a given threshold. One can consider a perceptron as a simple model for decision making by weighting up evidence. For example, your decision to go surfing may mainly depend on two factors, the strength of the wind (corresponding to input x_1) and how sunny it is (corresponding to input x_2). The weights represent your personal weighting between the two factors; choosing the weight $w_1 \gg w_2$ means that it is more important to have strong wind than to have sunshine. The bias b then sets your decision boundary and the outcome 1 corresponds to the decision to go surfing, whereas the outcome zero means that you will not go surfing. Clearly, this is a very simple decision model, but we will show in Sect. 3.3 that building networks of many neurons allows to represent complex models.

3.2 Activation Functions

The artificial neural networks currently used in practice use different activation functions for their neurons. Unfortunately, there is no theory that tells us which activation function works best for a certain machine learning task. However, as we will see in

Sect. 3.5, to be able to train the neural network, it is important that we choose a (piecewise) smooth activation function. Further, choosing a smooth activation function also allows a network to output real numbers instead of discrete ones, which is required for regression tasks in machine learning. Instead of using the discontinuous step function of a perceptron, we may use a smooth approximation of it, which is given by the sigmoid activation function (Fig. 3.1b), defined by

$$\sigma(z) = \frac{1}{1+e^{-z}}\,. \tag{3.2}$$

In practice, activation functions that do not saturate, i.e., that do not converge to a constant value for large inputs, usually perform better. Two commonly used activation functions are the Rectified Linear Unit (ReLU) [39, 40] and the Exponential Linear Unit (ELU) activation function [41], given by

$$\sigma_{\text{ReLU}}(z) = \begin{cases} z & \text{for } z > 0\,, \\ 0 & \text{for } z \le 0\,, \end{cases} \tag{3.3}$$

and, for some $\alpha > 0$

$$\sigma_{\text{ELU}}(z) = \begin{cases} z & \text{for } z > 0\,, \\ \alpha\,(e^{z} - 1) & \text{for } z \le 0\,. \end{cases} \tag{3.4}$$

The two functions are plotted in Fig. 3.2. For the implementation of the examples discussed in Part III of this book, we use the ELU activation function, apart from the example discussed in Sect.10.3, which uses the ReLU activation function.

3.3 Neural Networks

A (feed-forward) neural network is created by arranging neurons in layers and forwarding the outcomes of the neurons in the ith layer to neurons in the $(i+1)$th layer (see Fig. 3.3). The network as a whole can represent a class of functions $\{F_\theta\}_\theta$, parametrized by θ, which contains the weights and biases of all the neurons in the network. The inputs $x_1, \ldots, x_n$ of a function $F_\theta : \mathbb{R}^n \to \mathbb{R}^m$ in this class correspond to the activations of the neurons in the first layer (which is called the *input layer*). The activations of the input layer form the input for the second layer, which is a *hidden layer* (since it is neither an input nor an output layer). In the case of a fully connected network, each neuron in the $(i+1)$th layer receives the activations of all neurons in the ith layer as input. The activations of the m neurons in the last layer, which is called *output layer*, are then interpreted as the output of the function F_θ. Clearly, such

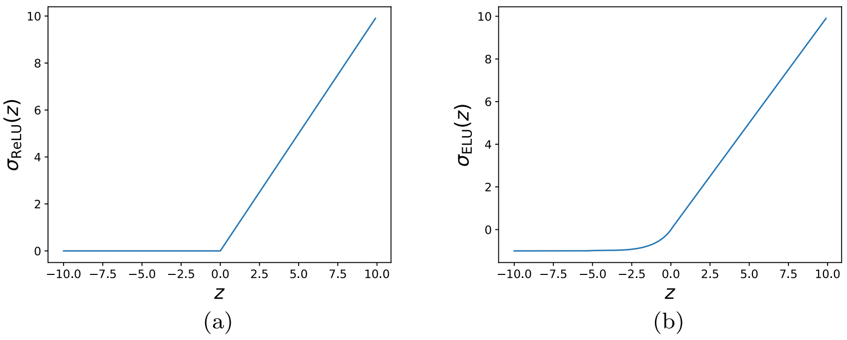


Fig. 3.2 ReLU and ELU activation functions. **a** Rectified Linear Unit (ReLU) [39, 40] given in (3.3). The ReLU activation function is linear for an input $z > 0$ and constantly zero otherwise. **b** Exponential Linear Unit (ELU) [41] given in (3.4) for $\alpha = 1$. The ELU activation function can be seen as a smooth version of the ReLU activation function (vertically shifted by –1)

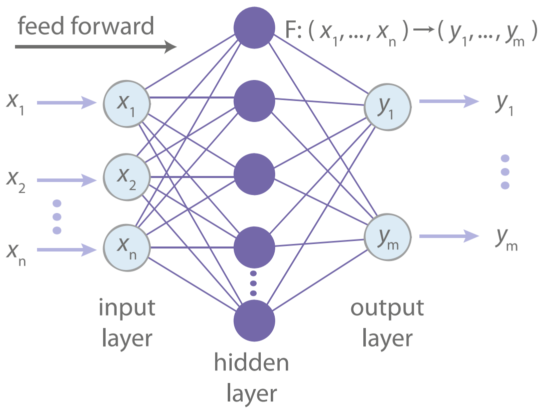


Fig. 3.3 Feedforward neural network. (Figure reproduced from Iten and Metger et al., *Physical Review Letters*, 2020 [22]). Fully connected (feed-forward) neural network with 3 layers. The network as a whole can be thought of as a function mapping the inputs $(x_1, \dots, x_n)$ to the output $(y_1, \dots, y_m)$. The layers consists of a series of artificial neurons (see Fig. 3.1)

networks can represent much more complex models than a single neuron. In fact, as discussed in the next section, for certain classes of activation functions, neural networks are *universal* in the sense that any continuous function can be approximated arbitrarily well by a sufficiently large feedforward network for a certain choice of the the weights and biases of the neurons in the network.

3.4 Universality Theorem

In this section, we investigate the expressiveness of neural networks, i.e., we investigate the class of functions that can be represented by neural networks of variable size. There are two classes of universality theorems. The first class considers the expressivity of neural networks for a fixed depth, i.e., a fixed number of hidden layers.

The second class considers the expressivity for fixed width, i.e. for a fixed number of neurons in each hidden layer. For this book, only the first class of universality theorems is relevant, and we refer to [42] for the second class.

A general result for the expressivity of neural networks with only one hidden layer was developed in [43, 44]. A version of the general universality theorem given in [44], which is particularly relevant for this book, is stated in the following theorem.

Theorem 3.1 *For any continuous, bounded and non-constant activation function, let us denote the class of functions that can be represented by a feed-forward neural network with one hidden layer and* p *hidden neurons by* $\{F_\theta^p : \mathbb{R}^n \to \mathbb{R}^m\}$. *Then, for any continuous function* $G : \mathbb{R}^n \to \mathbb{R}^m$ *and compact subset* $X \subset \mathbb{R}^n$ *and* $\epsilon > 0$, *there exists* $p \in \mathbb{N}$ *and a function* F_θ^p *such that* $\sup_{x \in X} \left| F_\theta^p(x) - G(x) \right| < \varepsilon$.

Hence, since the sigmoid activation function is continuous, bounded and non-constant, any continuous function can be approximates arbitrary well by a feed-forward network with a sufficiently large hidden layer containing sigmoid neurons. We refer to [44] for a proof of the statement and to [37] for an intuitive proof using visualization. The theorem does not apply to networks using the ReLU or ELU activation function, since they are not bounded. Nonetheless, a result established in 2017 [45] shows that neural networks with non-bounded and non-polynomial activation functions are also universal.

3.5 Training of Neural Networks

Even if we would know the target function that a neural network should implement, tuning the weights and the biases of all the neurons in the network by hand, such that the network well approximates the target function, would be a tedious task. Furthermore, using neural networks as a tool for machine learning, we would like them to learn a function themselves from training samples, i.e., known input-output pairs (x, y) with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. Hence, the goal is to find a training algorithm that takes a training set $\mathcal{S}$ consisting of input-output pairs as an input and adapts the biases and weights, summarized in a variable θ, of the neural network such that the function F_θ implemented by the network satisfies:

1. **Fitting**: $F_\theta(x) \approx y$ for all training input-output-pairs (x, y),
2. **Generalization**: $F_\theta(x') \approx y'$ for all test input-output-pairs (x', y') that has not been observed during training.

You may recall the example discussed in Sect. 2.1, where the machine learning task is to classify images of dogs and cats. The learning algorithm is thereby fed with input-output-pairs (x, y) consisting of images of cats and dogs (corresponding to x) and the correct label “cat” or “dog” (corresponding to y). More concretely, the input x to the neural network could be the values of the pixels of the images, and the output $F_\theta(x) \in [0, 1]$ could be the activation of a sigmoid neuron, which can be interpreted as the probability of having a dog on the image.[1] For example, if we get an outcome of 0.9 from the network, it is pretty sure that there is a dog on the image. The classification can then be done by classifying the image as "dog" if $F_\theta(x) \geq 0.5$ and as a cat otherwise. The optimal output would be $F_\theta(x) = y$, where we set $y = 1$ if the label is "dog" and $y = 0$ if the label is "cat". The test data then checks how well images that have not been seen during training are classified.

Let us first focus on a training algorithm to achieve goal 1. To train a neural network, we need some performance measure of the network that smoothly depends on the network parameters θ (i.e., the biases and weights of the network). In the example above, such a measure could be chosen as $C((x, y), \theta) = (F_\theta(x) - y)^2$ for a training sample (x, y). We call $C((x, y), \theta) \geq 0$ the *cost function*. The name is motivated by the fact that we would like to minimize the cost, i.e., we would like to have $C((x, y), \theta) \approx 0$ for all training examples (x, y), which corresponds to goal 1. The performance of the network over the whole training set is then measured by the average cost defined by

$$\bar{C}(\theta, \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{(x,y)\in\mathcal{S}} C((x, y), \theta) \ , \tag{3.5}$$

where $|\mathcal{S}|$ denotes the number of samples (x, y) in the training set. Note that the averaged cost function is a sum of smooth functions in θ, and hence, depends itself smoothly on θ.

3.5.1 Stochastic Gradient Descent

The goal of the training algorithm for neural networks is to find a choice for the weights and biases in the network, such that the average cost given in (3.5) is minimized. The basic idea is to start from randomly chosen weights and biases and then stepwise adapt them to lower the average cost. In every step, we change each of the weights and biases a little bit, i.e., $w_i \to w_i + \delta w_i$ and $b_i \to b_i + \delta b_i$ for some small $\delta w_i > 0$ and $\delta b_i > 0$, and observe the effect of the change on the average cost. Then, we update the weights and biases for which the cost decreases by $w_i \to w_i + \delta w_i$ and $b_i \to b_i + \delta b_i$, and the ones for which the cost increases by $w_j \to w_j - \delta w_j$ and $b_j \to b_j - \delta b_j$. For small enough δw_j and δb_j, the average cost is decreased by these updates.

In practice, instead of working with small changes, one calculates the gradient of the cost function with respect to all weights and biases, which indicates the direction

[1] The hidden structure of the networks that are commonly used for image classification are not fully connected, but are so called *convolutional neural networks* (see for example [37] for an introduction), which are optimized for processing visual data. However, this is irrelevant for the conceptual discussion here.

in which the cost function maximally increases (in an infinitesimal neighborhood). Moreover, it is usually computationally too expensive to calculate the change of the average cost function, since one has to calculate the change separately for all training samples and usually, the training set is quite large. Hence, instead one estimates the change of the cost function by averaging only over a part of the training set, called a *mini-batch*. More concretely, the training algorithm works as follows (using again the vector θ containing all weights and biases of the network as elements to simplify the notation):

1. Initialize the weights and biases randomly.
2. For $t \in \{1, 2, \ldots, T\}$, where T denotes the number of training steps, repeat the steps 3 and 4.
3. Randomly choose a subset (called mini-batch) $\mathcal{M}_t \subset \mathcal{S}$ consisting of $k \in \mathbb{N}$ samples.
4. Update the network parameters by

$$\theta_t \to \theta_{t+1} = \theta_t - \eta \nabla_\theta \bar{C}\left(\theta_t, \mathcal{M}_t\right) \ , \tag{3.6}$$

for some learning rate $\eta > 0$.

This learning algorithm is called *stochastic gradient descent*. The name comes from the fact that the mini-batch is chosen randomly leading to a stochastic estimate of the gradient. The gradients can be efficiently calculated by the *backpropagation algorithm* that essentially calculates the gradients of the parameters of the neurons at the output layer, and then subsequently applies the chain rule for derivatives to calculate the gradients for earlier layers until ending up with the input layer (the interested reader can find more details in [37]). The mini-batch size k and the learning rate η are so called *hyperparameters*. Hence, they are not adapted by the algorithm, but have to be chosen by hand before running the algorithm. The bigger we choose the mini-batch size, the better the estimate of the gradient gets in average in tradeoff with larger computation time for each update step. The learning rate determines the size of $\delta\theta$, i.e., how much we update the network parameters in each step.

3.5.2 Convergence and Choice of Hyperparameters

Generally, given an optimization problem and some hyperparameters, the goal is to choose the hyperparameters in a way that maximizes the convergence speed. However, for cost functions that are non-convex in θ, which is commonly the case for neural networks, we have no guarantee that stochastic gradient descent converges to the global minimum of the cost function. Indeed, usually it will get stuck in a local minima, and it depends on the random initialization of the network parameters, in which local minima we get stuck. Nonetheless, in practice, training a network several times with random initialized parameters θ and then choosing the network with the best performance works well for a lot of problems. Unfortunately, it is not well understood yet why this is the case and it is an active field of research to better understand the training of neural networks. Furthermore, we do also not have a theory that tells us the optimal choice of the hyperparameters to optimize the convergence speed or alternatively to optimize the probability to end up in a local minima that achieve prediction accuracy close to the optimal minima.

Therefore, choosing good hyperparameters and random initialization strategies are currently based on experience and some intuition and tricks. A comprehensible overview of the current state of the art is given in [37], but it is beyond the scope of this introduction into neural networks to provide an extensive overview here. In the following, we just describe the essential difficulty in choosing good hyperparameters. A priori, one may think that choosing a large mini-batch size and a small learning rate will improve the performance of the trained network in tradeoff with computational time. Unfortunately, things are more complicated with neural networks. Basically, the learning rate and the mini-batch size should be chosen in a way to achieve a good tradeoff between (i) converging fast to a local minima, and (ii) missing out local minima with bad performance. Indeed, choosing a small mini-batch size introduces some stochasticity in the updates of the network parameters, which may at first hand seem like something one would like to avoid since it temporarily may increase the average cost if the mini-batch is too small to be representative. However, it turns out that a certain amount of such stochasticity can help to jump out of local minima with bad performance, and hence to converge to another local minima with better performance. Similarly, choosing a small learning rate ensures that we converge to the closest local minima. On the other hand, choosing a large learning rate may lead to an "overshooting" of a local minima, i.e., the average cost over the full training set may actually increase in such a training step. Again, this could help to avoid bad local minima. On the other hand, choosing a too small mini-batch size and a too large learning rate will essentially lead to a random walk in the cost landscape and the network may not be able to improve its performance at all. To conclude, training neural networks and choosing the optimal hyperparameters is currently more based on experience than rigorous insights and requires some patience in trying out different choices of hyperparameters until a good choice is found.

3.5.3 Generalization

A neural network of moderate size consists of a large amount of parameters. For example, a fully connected network with 10 input- and output-neurons, and a hidden layer consisting of 100 neurons, contains 120 biases and $2 \cdot 10 \cdot 100$ weights, which are over 2000 parameters in total. According to the bias-variance tradeoff discussed in Sect. 2.4, we would expect such a complex model to overfit and generalize badly due to high variance. In other words, we would expect that the training data can be fitted well, but that the generalization power of a neural network is very bad.

Surprisingly, using some tricks for regularization, neural networks show amazing generalization power for tasks like image classifications.[2] However, to achieve good generalization power, the choice of the regularization technique is essential and, again, we do not have a deep understanding yet of which regularization methods should be used for a given setting. One regularization method is for example to add a term $\|\theta\|_1$ to the cost function to motivate the network keeping parameters small and setting unnecessary parameters equal to zero. Intuitively, motivating the network to reduce the model complexity by setting unnecessary parameters to zero lowers the variance of the model and hence may increase the generalization power. For a further discussion on state of the art techniques for regularizing neural networks we again refer to [37].

3.6 Deep Learning

Very likely, you have already heard about deep learning. So what is it exactly? Essentially, training any neural network with at least one hidden layer can be considered to be deep learning. In practice, the term is often used for networks containing many hidden layers (quite common are numbers of around five to ten hidden layers). Recalling from Sect. 3.4 that a neural network with one hidden layer can approximate any continuous function with sufficiently many hidden neurons, one may wonder why we should even consider networks with more than one hidden layer. Just because something is possible does not mean it is efficient, in the sense that with multiple layers, fewer neurons may be needed overall to achieve the same expressiveness as a network with one hidden layer (see e.g. [46] for a mathematical comparison of the expressiveness of shallow and deep ReLU networks). Furthermore, in principle, it could be the case that deep neural networks have better generalization properties than shallow ones.

Intuitively, there is a compositional argument why deep learning works well in praxis. Recall again the task of classifying images of cats and dogs considered in Sect. 2.1. As demonstrated in [47, 48], neural networks that are trained to classify images store gradually more complex features in later layers. While the first few layers may represent simple features like edges of different orientation, later layers can build on these features to form more complex part-of-object shapes like typical eyes of dogs or cats. The part-of-object shapes can then be combined further to make a decision on the classification task and output the label “cat” or “dog”.

In praxis, deep neural networks often outperform shallow networks for tasks like image recognition. But unfortunately, deep networks tend to be difficult to train in praxis. While deep networks are also trained using gradient descent (and the

[2] For image classifications, one does not use fully connected neural networks, but so called *convolutional neural networks*. These networks use some prior knowledge about “locality” of information in images and translation invariance of visual features. Nonetheless, the number of parameters in such networks is still significantly larger compared to more conventional machine learning models.

backpropagation algorithm is used for efficiently calculating the gradients), the size of the gradients turn out to behave unstable in such networks. Hence, the gradients in the earlier layers of such networks tend to explode (*exploding gradient problem*) or to nearly vanish (*vanishing gradient problem*). We refer again to [37] for a discussion why this problem appears and how it is handled in praxis.

Sometimes one may not only be interested in the classification of pictures of dogs and cats, but also in extracting relevant features of cats and dogs from images. Finding a compact representation of data by extracting high level features is exactly the goal of representation learning discussed on the basis of autoencoders in Chap. 4. Deep neural networks are a powerful tool for representation learning, since they quite naturally learn high level features in deeper layers.

Chapter 4
Autoencoders

Autoencoders are a tool for representation learning, which is a subfield of unsupervised machine learning and deals with feature detection in raw data. A well known example for representation learning is PCA, discussed in Sect. 2.2. The most methods that are currently used for representation learning are based on artificial neural networks. While, in principle, all deep neural network architectures learn some representation within their hidden layers (see also Sect. 3.6), most work in representation learning is dedicated to defining and finding *good* representations [27]. The main focus of Part III of this book is the formalization of what a *good* representation is for physical systems and designing neural network structures that are able to find such representations. While there are a variety of tools for representation learning [27], the one most relevant for this book is the autoencoder.

In the following, we provide a non-technical introduction to autoencoders. A mathematical discussion of a specific kind of autoencoders (so called *variational autoencoders*) is given in Appendix B. An autoencoder is a tool to find a compact representation of samples in a data set. For concreteness, think about a data set containing images of circles of different radius and centered at different points in the s-t-plane (see Fig. 4.1). As discussed in Sect. 3.6, deep neural networks are able to extract abstract features from data. However, so far, we have only considered neural networks for supervised learning. Here, we only have given images of circles, but we do not have any labels, so what should the output of the neural network be? As we search the hidden layers of the network for representations that still contain enough information to reconstruct the image, we should ask the network to reproduce the input image again as the output of the network, i.e., for an image x and a network F_θ we want $F_\theta(x) \approx x$. But this is just the identity function, so what is the point of training a network to learn the identity function? The network becomes interesting by introducing a dimension "bottleneck" in a hidden layer, i.e., by introducing a hidden layer with a small number of neurons (called *latent layer*). This forces the network to compress the input to a few relevant parameters and does not allow it to just successively pass the original image from one layer to the next. More concretely, we can split the network into two parts, the *encoder* function $E(x) = r$ mapping an input x to a latent representation $r \in \mathbb{R}^p$ with p latent neurons and where the elements

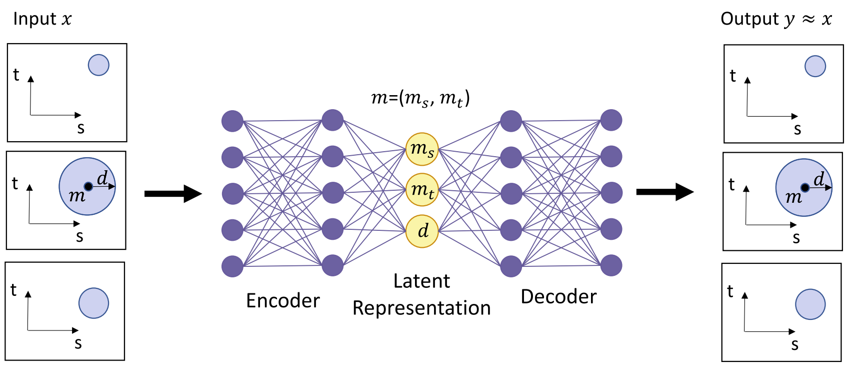


Fig. 4.1 Autoencoder. The figure shows an autoencoder used to compress images of circles. Three possible input images are shown. The position as well as the radius of the circle varies over the training data. An encoder mapping is used to compress the image data to a small dimensional latent representation. The high dimensional input to the encoder may corresponds to the values of the pixels of the image, and the output of the encoder is given by the activations of the neurons in the latent layer. Note that the number of neurons depicted for the encoder (and decoder) is not representative. An efficient encoder recognizes that it is sufficient to "store" the coordinates $m = (m_s, m_t)$ of the center of the circle together with its radius d. We would like to stress that these values are not parameters of the latent neurons, but correspond to the real valued output of theses neurons (given an image as an input to the encoder). A decoder is then used to reconstruct the image using only the information from the latent representation

of r correspond to the activation of the latent neurons for the given input x. The latent representation r is then mapped by the second half of the network, the *decoder* function $D(r) = y$, to the output y of the network (which should be approximately equal to the input image, i.e., $x \approx y$). In the considered example (Fig. 4.1), knowing the centre (m_s, m_t) of the circle and its radius d is sufficient to reconstruct the image. Hence, the input image, which might be quite high dimensional, e.g., a vector with 10,000 entries if the image dimension is 100×100 pixels, can be compressed to a three-dimensional latent representation storing m_s, m_t and d. Note that features that stay the same over all images, such as the shape (in this case a circle), are "stored" in the decoder. The latent representation only has to store the information that may change from one image to another.

Where it seems most natural as a human to store m_s, m_t and d in a latent representation consisting of three neurons, there is no a priori reason for the neural network not to store e.g. $a = m_s + m_t$, $b = m_s - m_t$ and $c = d$, since the decoding function can easily recover $m_s = (a + b)/2$, $m_t = (a - b)/2$ and $d = c$ from this representation. Therefore, in resulting representations, different parameters stored in the latent neurons are often highly correlated and do not have a straightforward interpretation. A lot of work in representation learning has recently been devoted to *disentangling* such representations, i.e., to seperate the parameters in the representation in a meaningful way (see e.g. [49–53]). In particular, these works introduce criteria, also referred to as *priors* in representation learning, by which we can disentangle representations. Where several work focusses on a prior of having statistically independent latent neurons, a recently proposed prior is the so called *consciousness prior* [54]. It suggests to disentangle the latent representation via an attention mechanism by assuming that, at any given time, only a small fraction of the features or concepts stored in the representation are sufficient to make a useful statement about reality. In Part III of this book, we discuss criteria to disentangle representations of physical system in a natural way.

Part II
Overview of Using Machine Learning for Scientific Discoveries

In this part, we give an overview of different directions in which machine learning can be applied to automate different aspects of the scientific discovery process. In recent years, machine learning has become a common tool for science, and it is beyond the scope of this part of the book to discuss all related results. Rather, we focus on a selection of results with the following characteristics: (i) they are based on machine learning methods that can be applied to a wide range of physical systems, i.e., the methods are not specialized to a particular class of systems; (ii) they are representative of a research direction (and hence, they provide the reader with the necessary machine learning background to read the latest results in the corresponding research field); and (iii) they allow the extraction of conceptual information from the physical system. Each chapter in this part of the book is self-contained, and hence, the reader is free to pick any chapters he or she is particularly interested in, and skip others if they are less relevant to his or her field of study.

The most methods discussed in this part require the knowledge about which physical parameters are relevant to describe the system at hand. For example, it may be assumed that the relevant parameters to describe the time evolution of a charged particle in a homogeneous electric field are the charge and the mass of the particle, as well as the field strength. Recent advances in deep learning and representation learning allow to get rid of this prior knowledge and to let the machine discover the relevant parameters itself. This is discussed in Part III of this book.

Chapter 5
Creating Experimental Setups

As discussed in Sect. 1.2, creating experimental setups is a fundamental step in a physicist's discovery process. Creating experimental setups for quantum systems is particularly challenging, since the behavior of such systems is often unintuitive. In the last view years, automated search techniques and reinforcement-learning based schemes have been used to generate new experimental setups for quantum optics [55, 56] (see [57] for a review of computer inspired experiments). This research has lead to the solution of several previously unsolved questions [55, 58], experimental implementation of computer inspired experiments [59–64] and last but not least, to the discovery of new scientific ideas and concepts [65–67]. Recently, a new design algorithm named THESEUS was introduced in [68] specifically aiming at conceptual insights. THESEUS uses graph representations fo quantum experiments [66], which significantly speeds up the runtime compared to the automated search algorithm MELVIN [55]. Instead of completely automatizing the generation of experiments, one may alternatively also use machine learning to generate hypotheses which can help to improve a human expert's intuition about the system [69].

In this chapter, we focus on the results described in [56], which are based on a physically motivated kind of reinforcement learning, called projective simulation [70]. Projective simulation is particularly well suited for reinforcement learning problems where one wants to gain some insight into the thinking process of the agent. Indeed, projective simulation allows to discover combinations (or *gadgets*) of experimental tools that turn out to be useful outside of the given task of the agent. Hence, the following discussion can be considered in a broader context as providing an example of how to gain conceptual insights into the interaction of an artificial physicist with its environment using reinforcement learning.

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_5

5.1 Problem Setting from Quantum Optics

In this section, we provide the background required to understand the quantum optics setup discussed in the following sections (and based on [56]). It is not necessary to have an understanding of quantum mechanics to follow this section. Enough background knowledge is provided so that the considered goal of generating high-dimensional entangled quantum states can be regarded as a purely mathematical optimisation problem.

A (pure) state of a d-dimensional quantum system can be represented as a complex vector $|\psi\rangle \in \mathbb{C}^d$ that is normalized, i.e., with unit Euclidean norm $\||\psi\rangle\|_2 = 1$, and where two states $|\psi\rangle$ and $|\psi'\rangle$ are identified if and only if they differ by a global phase factor, i.e., if there exists $\phi \in \mathbb{R}$ such that $|\psi\rangle = e^{i\phi}|\psi'\rangle$. We work with the standard inner product on the complex vector space, which is given by $\langle v|w\rangle = \sum_{i=1}^{d} v_i w_i^*$, for two d-dimensional vectors $v_i, w_i \in \mathbb{C}^d$ and where the $*$-symbol denotes the complex conjugate. The vector space $\mathbb{C}^d$ together with the standard inner product is then called a *Hilbert space*.[1]

In the quantum optics setup used in [56], the position and the orbital angular momentum (OAM) of photons are used as the degrees of freedom to store and manipulate quantum states. We work with four different paths, labeled by a, b, c and d where the photons can travel, and with $2M + 1$ angular momentum states of each photon, labeled with $-M, \dots, 0, \dots, M$, and where $M \in \mathbb{N}$ is some finite upper bound on the angular momentum number m of a photon. Hence, the state of a photon lives in a d-dimensional complex Hilbert space with $d = 4 \times (2M + 1)$ and orthogonal basis vectors $|m\rangle_p$ with $m \in \{-M, \dots, 0, \dots, M\}$ and $p \in \{a, b, c, d\}$.

The goal considered in [56] is to create a special kind of photon states, so called *entangled* states. Before we (mathematically) describe the experimental tools that are available to achieve this goal, let us discuss the definition of entanglement. Entanglement plays a fundamental role in various fields of research that are based on quantum mechanics. It describes an intrinsic property of quantum systems that forms the basis for the conceptually challenging behavior of such systems [71] and finds many applications in quantum information theory (see [72] for an overview) and in condensed matter theory, where a strong connection between phase transitions in complex systems and entanglement was discovered [73]. Let us first define entanglement for bipartite states, and consider the more evolved case of many parties afterwards.

5.1.1 Entanglement of Bipartite Systems

Consider two quantum systems A and B whose states live in Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$ of dimensions d_A and d_B, respectively. The state of the composite system

[1] A Hilbert space is a real or complex inner product space that is a complete metric space with respect to the distance function induced by the inner product. The distance function, induced by an inner product $\langle\cdot|\cdot\rangle$, between two vectors x and y is defined by $\langle x - y|x - y\rangle$.

then lives in the tensor product of the Hilbert spaces $\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B$ (see [72] for a detailed introduction to tensor products in the context of quantum mechanics). For example, if the system A and B are in states $|\psi_A\rangle$ and $|\phi_B\rangle$, respectively, the composite state is given by $|\psi_A\rangle \otimes |\phi_B\rangle$. States that can be written in tensor product form are called *separable* or *product states*. Maybe surprisingly if one is not already familiar with quantum mechanics, not all states in $\mathcal{H}_{AB}$ are product states, hence, there are states that can not be described by two local states of each of the systems. Rather, if the system is in a non-separable state (also called *entangled state*), the two systems form an inseparable whole. An example for a state of two photons at path a and b that is (maximally) entangled in the OAM basis is give by

$$|\psi_{AB}\rangle = \frac{1}{\sqrt{3}} \sum_{m=-1}^{1} |m\rangle_a |-m\rangle_b \,. \tag{5.1}$$

Note that the factor $1/\sqrt{3}$ is required, such that $\langle\psi_{AB}|\psi_{AB}\rangle = 1$ is normalized.[2] Such states can be experimentally generated by a double spontaneous parametric down-conversion process in two nonlinear crystals [55, 59]. A way to characterize entanglement is based on the Schmidt decomposition.

Theorem 1 (Schmidt decomposition [72]) *Let $|\psi_{AB}\rangle$ be a composite state of two systems A and B. Then, there exists orthonormal states $|\psi_i\rangle_A$ for system A, and orthonormal states $|\phi_i\rangle_B$ for system B, such that*

$$|\psi_{AB}\rangle = \sum_i \lambda_i |\psi_i\rangle_A |\phi_i\rangle_B \,, \tag{5.2}$$

where λ_i are non-negative real numbers satisfying $\sum_i \lambda_i^2 = 1$ and known as the Schmidt co-efficients.

The proof of the decomposition is based on the singular value decomposition and is given in [72]. Apart from many other applications of the Schmidt decomposition in quantum information theory, the minimal number of Schmidt coefficients, called the *Schmidt rank*, can be used to characterize the entanglement by the minimal number of required degrees of freedom of the local systems that are entangled with each other. A product state requires only one Schmidt coefficient and hence has a Schmidt rank equal to one. This corresponds to the interpretation that the system is "entangled" over one degree of freedom, which is equivalent to say that each local system is independent of the other one and hence the state is separable. On the other hand, any state with Schmidt rank bigger than one is entangled. For example, one can show that the state given in (5.1) has Schmidt rank three, hence it is entangled in

[2] The notation $\langle\psi_{AB}|\psi_{AB}\rangle$ abbreviates the expression $\langle|\psi_{AB}\rangle||\psi_{AB}\rangle\rangle$ and is actually one of the underlying reasons why it is common to bracket a quantum state by $|\cdot\rangle$, which is called the "ket"-notation. Together with the "bra"-notation $\langle\psi_{AB}|$, denoting the complex conjugate row vector of the column vector ψ_{AB}, the "braket"-notation $\langle\psi_{AB}|\psi_{AB}\rangle$ can be read as the matrix multiplication of $\langle\psi_{AB}|$ with $|\psi_{AB}\rangle$.

all three OAM dimensions. Note that this characterization does not depend on the size of the Schmidt coefficients, and hence is not robust against noise: an arbitrary small disturbance of the state may change the Schmidt rank e.g. from one to two and hence the characterization from “separable” to “entangled”. Nevertheless, this characterization works well in the context considered in the following sections, in particular since the quantum systems used in [56] are simulated on a computer with arbitrary accuracy.

5.1.2 Entanglement of Multipartite Systems

Characterizing the entanglement of multipartite systems becomes much more involved than for bipartite systems. To demonstrate this, let us consider a tripartite system ABC. We can characterize the entanglement between the system $A - BC$, $B - AC$, and $C - AB$ separately by the Schmidt rank. However, in general, the three Schmidt ranks are not equal, and hence we have not a single number characterizing the entanglement, but three numbers (which are not fully independent, but also not determined by each other [74]). Hence, we may define the Schmidt rank vector (SRV) of a tripartite system as a three dimensional vector with integer entries corresponding to the Schmidt ranks of the different bipartite splittings of the system. The entries of the SRV can be considered as the *entanglement dimensions* between the different splittings of the system. The goal considered in [56] is to create high-dimensional entangled tripartite photon states with SRV (3, 3, 2) and (3, 3, 3). Such states are not only of fundamental interest, but also because of the many applications in quantum communication and computation [75–80].

5.1.3 Preparation of Photon States

Initially, two photon pairs are created that are maximally entangled in the OAM basis, i.e., we start from a state of four photons given by

$$|\psi\rangle = \frac{1}{3}\left(\sum_{m=-1}^{1} |m\rangle_a | - m\rangle_b\right) \otimes \left(\sum_{m=-1}^{1} |m\rangle_c | - m\rangle_d\right) . \tag{5.3}$$

Then, we can apply different transformation to this state corresponding to well known tools from quantum optics with the goal of creating high-dimensional entanglement between three of the photons. The available tools together with their mathematical description are listed below.

1. $\mathrm{BS}_{a,b}$: Nonpolarizing symmetric 50/50 beamsplitter for photons on path a and b:

$$|m\rangle_a \rightarrow \frac{1}{\sqrt{2}}\left(i|-m\rangle_a + |m\rangle_b\right) , \tag{5.4}$$

$$|m\rangle_b \rightarrow \frac{1}{\sqrt{2}}\left(|m\rangle_a + i|-m\rangle_b\right) . \tag{5.5}$$

2. $\text{Holo}_{a,k}$: Hologram in path a with a OAM shift $k \in \{\pm 1, \pm 2\}$:

$$|m\rangle_a \rightarrow |m+k\rangle_a . \tag{5.6}$$

3. DP_a: Dove prism on path a:

$$|m\rangle_a \rightarrow ie^{i\pi m}|-m\rangle_a . \tag{5.7}$$

4. Refl_a: Mirror on path a:

$$|m\rangle_a \rightarrow |-m\rangle_a . \tag{5.8}$$

The goal is, starting from the state $|\psi\rangle$ given in (5.3), to apply a sequence of the transformations listed in 1 to 4, such that we end up with a high-dimensional entangled state on three photons. After applying all transformations, the photon in path a is post-selected in the OAM basis meaning that it is measured in the OAM basis and that for all possible measurement outcomes, the states of the remaining three photons are investigated. For example, if we would not apply any transformation on the initial state $|\psi\rangle$, postselecting in the OAM basis would lead to three non-trivial state corresponding to postselection on $|m_0\rangle_a$ with $m_0 \in \{-1, 0, 1\}$. Dependent on the measurement result m_0 of the orbital angular momentum of the photon in path a, we would obtain the following state on the remaining three photons

$$\frac{1}{\sqrt{3}}|-m_0\rangle_b \otimes \left(\sum_{m=-1}^{1} |m\rangle_c|-m\rangle_d\right) . \tag{5.9}$$

This state has a SRV of (1, 3, 3), hence, the photon in path b is not entangled at all. The goal is to find a sequence of transformations to end up with a three photon state with SRV (3, 3, 2) or (3, 3, 3). This is a quite challenging task and it is discussed in the next section how reinforcement learning can be used to tackle it.

5.2 Creating Experimental Setups Using Projective Simulation

Projective simulation (PS) is a specific method for reinforcement learning introduced in 2012 in [70]. PS has shown to perform well for standard reinforcement learning problems [81–84], advanced robotics problems [85, 86], quantum error correction [87, 88], quantum communication [89], setting up experimental Bell tests [90], modelling collective behavior based on the principle of agency [91–93], and even turned out to allow for quantum enhancements for reinforcement learning [94–100]. Furthermore, some interesting philosophical challenges of agency were considered in the context of projective simulation [101–103]. Recall from Sect. 2.3 that the goal of an agent in reinforcement learning is to process an input encoding the state of the environment to decide which action it should perform next to achieve a goal. One of the fundamental ideas behind PS is that the agent does the processing of an input by projecting itself, based on previous experiences and variations thereof, to potential future situations. Based on the associated situations (and the current state of the environment given as input), the agent chooses the next action to perform on the environment. This idea is related to episodic memory whose role for planning and prediction was already discussed in psychology in the 1970s [104, 105].

Let us informally explain the idea on the basis of a simple example (discussed formally in [70]). Consider an agent that can decide to choose actions a_r or a_l in each time step, meaning that he moves to the right or left, respectively. The input to the agent is a red arrow pointing to the left s_l^{red} or to the right s_r^{red}, and the correct movement of the agent should be according to the shown arrow. The PS agent learns to associate the shown arrow with the correct actions, i.e., it associate the action a_r with s_r and a_l with s_l. Now, let's assume that the arrow shown to the agent is sometimes colored blue (and the other times it is still colored red), hence there are two additional potential inputs to the agent denoted by s_l^{blue} and s_r^{blue} corresponding to the blue arrow pointing left and right, respectively. Instead of learning to associate the blue errors again with the correct actions, the PS agent could also learn to associate the blue and red arrow pointing in the same direction with each other, hence recognizing that the "meaning" of the arrow stays the same independent of the color. The memory episode would then look like this, for example given an input s_l^{blue}, this input would activate s_l^{red}, which in turn would lead to choosing the correct action a_l. The reflection is hence not realized as a sophisticated computational process, but it can be seen as a structural-dynamical feature of the memory itself. In particular, this allow to generalize the idea of PS to the quantum world as discussed in [70]. The elementary excitations of the episodic memory are called *clips*, which are given by s_l^{blue}, s_l^{red}, s_r^{blue}, s_r^{red}, a_l and a_r in the considered example.

The discussed example is very simplistic and ignores several aspect of the general framework of PS [70]. In the general framework the associations between different clips are probabilistic. PS allows to reason about various potential situations by performing a random walk in the clip network without applying any action that is exited during this process to the environment. Hence, it is only reasoned about the actions in the first place, but they are not directly implemented in "reality". Finally, dependent on the whole reasoning process, a "real" action is chosen to perform on the environment. Importantly, clips can not only consist of one state or action, but also of a composition. PS is able to learn such compositions on its own, which is a key feature helping to extract conceptual information from the trained PS agent. This idea will be explained in detail in the concrete example from quantum optics (Sect. 5.1) in the following.

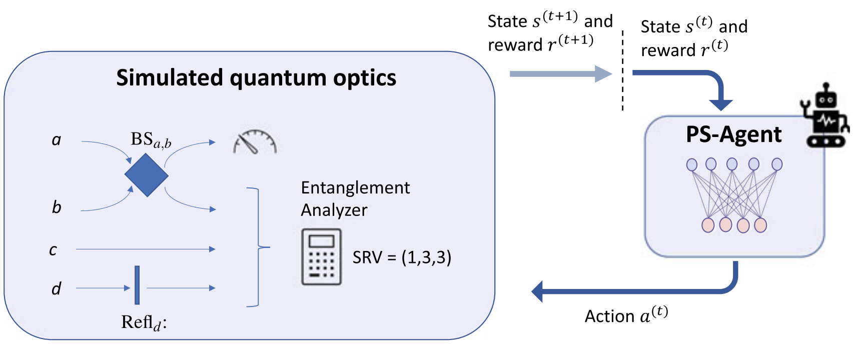


Fig. 5.1 Environment for projective simulation agent [56]. A PS agent places in each step t (dependent on the current optical setup $s^{(t)}$) additional optical tools at some position on a (virtual) optical table. The action $a^{(t)}$ determines the number of tools and which tools are added at what place on the optical table. For example, the action $a^{(t-1)}$ could have been set to a beam splitter $\mathrm{BS}_{a,b}$ on path a and b (which is shown in current state $s^{(t)}$ in the image). The evolution of four photons propagating on four paths a, b, c and d is simulated by a computer and the output state on path b, c, d (postselected in the OAM basis of the photon on path a) is calculated. Then, the Schmidt-Rank vector (SRV) of the output state is determined as an entanglement measure. The agent obtains a positive reward $r^{(t+1)}$ if the SRV of the three photon state is $(3, 3, 2)$ or $(3, 3, 3)$. If the reward is positive or if the maximal number L of optical elements is achieved, the experiment ends. Otherwise, the agent choses the next elements to place on the optical table dependent on state $s^{(t+1)}$

5.2.1 Architecture for PS Agent

The PS agent used in [56] has a simple architecture, where each remembered state of the optical setup and each action corresponds to a single clip. The PS agent builds up the optical setup stepwise, where in each step it places a new optical element on the optical table (Fig. 5.1). Hence, the set of $K \in \mathbb{N}$ possible actions $\{a_i\}_{i\in\{1,\ldots,K\}}$ correspond to the possible transformations listed in 1–4. We have 6 possible ways to place a beam splitter (choosing two paths out of four), 16 different variants of Holograms, and 4 choices for Dove prisms as well as for reflections summing up to $K = 30$ possible actions in total. The clips corresponding to the optical setups are added to the architecture on the go, i.e., starting from an architecture with state clips $\{s_i\}_{i\in\{1,\ldots,N\}}$, each created setup that is not already contained as a clip s_i in the architecture is added as as an additional clip s_{N+1}.

We are interested in finite experiments, so the maximal number of optical elements in one experiment is limited to $L \in \mathbb{N}$. This is not solely a theoretical restriction, but rather of practical interest: since each optical element adds some noise to the quantum state, e.g. through misalignment and interferometric instability, we expect a decreasing fidelity of the resulting state for longer experiments. The number of possible states of the optical setup with a fixed number l of optical elements is given by K^l. And hence, the set of all possible states $\{s_i\}_{i\in\{1,\ldots,N_{\max}\}}$ consists of $N_{\max} = \sum_{l=0}^{L} K^l$ elements. Note that thereby, s_1 represents the empty optical table.

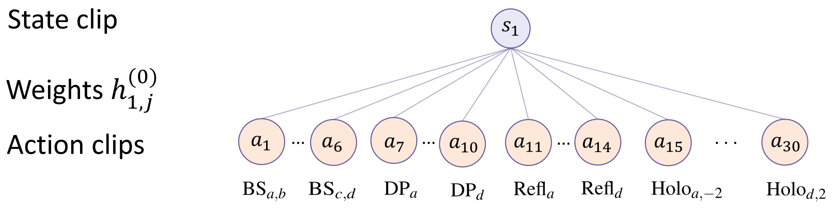


Fig. 5.2 Initial architecture for the PS agent [56]. The figure shows a simple model for the simulation of the episodic memory of the PS agent at initialization. The goal of the agent is to create experimental setups that create entangled photon states (see Fig. 5.1 for the setting). The states of the environment and the actions that the agent can choose correspond to single clips. The clip s_1 corresponds to the empty optical table. (Further clips corresponding to states of the agent are memorized on the go during the training of the agent.) The action clips $a_1, \ldots, a_{30}$ correspond to placing one of the possible tools listed in 1 to 4 on the optical table. The state s_1 is connected to all actions a_j via weighted edges $(1, j)$. The weights $h_{1,j}^{(0)} = 1$ are all initialized with one, and hence, the first action is chosen uniformly at random

Working with at most $L = 6$ optical elements, this already corresponds to a maximal number of clips of more than half a billion.

The initial architecture (which may be changed during training) for the PS agent then just consists of the clip s_1 corresponding to the empty optical table and of clips $a_1, \ldots, a_K$ corresponding to actions placing a single optical tool listed in 1 to 4 on the table (Fig. 5.2). The state s_1 is connected to every action a_j via a weighted edge $(1, j)$ with weight $h_{1,j}^{(0)}$, where the superscript denotes the number of passed training steps. An edge (i, j) represents the possibility of taking an action a_j in a situation s_i. More concretely, the probability $p_{i,j}^{(0)}$ of choosing action a_j given state s_i is proportional to the weight $h_{i,j}^{(0)}$, i.e. we have

$$p_{i,j}^{(0)} = \frac{h_{i,j}^{(0)}}{\sum_{k=1}^{K} h_{i,k}^{(0)}} . \tag{5.10}$$

Initially, all weights are set equal to one [56], and hence, the probability of performing action a_j starting from the empty optical table s_0 is uniform, i.e., $p_{1,j} = 1/K$ for every $j \in \{1, \ldots, K\}$. Similarly, if an additional state clip s_i is memorized during the training process, the weights for the edges (i, j) are initialized with one.

5.2.2 *Training of the PS Agent*

Starting from randomly generating optical setups by letting the PS agent choose which optical element to place next, we would like to train the PS agent to learn producing setups that lead to high-dimensional entangled states with high probability. In contrast to supervised learning, we have no knowledge about which actions the PS agent should choose to achieve this goal. The only feedback on its performance that the PS agent gets is a positive reward in the case he created an optical setup leading to a high-dimensional entangled state, or no reward if the maximal number of elements on the optical table is reached. In both cases, we reset the optical table afterwards to the empty state and start a new experiment where the PS agent again places optical tools on the table. From such rewards, the agent has to learn which elements to place next, given an optical table with some elements already placed. Three different kind of adaptions during the training allow the agent to adapt his behavior [56]:

1. Adapting the weights of the PS architecture, and hence, the probabilities to choose an action a_j given a state s_i of the optical table.
2. Action composition, i.e. adding compositions of several actions that have proven useful in achieving the goal as an action clip to the architecture.
3. Clip deletion, i.e., deleting states and actions that are likely not required to achieve the goal.

We discuss all three learning techniques separately in the following (readers not interested in the details how the learning works may safely skip Sects. 5.2.2.1 and 5.2.2.3). Action composition (Sect. 5.2.2.2) is in particularly interesting since it can help to get some conceptual insights into which gadgets, i.e., combination of experimental tools, are especially useful in combination.

5.2.2.1 Training of the Weights

The goal is to update the weights during the training of the PS agent to optimize the (average) probability of success, i.e., the chances of receiving a positive reward. The basic idea to achieve this is the following: each time the PS agent receives a reward, all the weights of the edges that correspond to the choices of actions made during building up the optical setup are increased. For example, assume that, starting from the empty optical table state s_1, the PS agent first performs an action a_{14}, leading to a state s_3 corresponding to an optical table with a single mirror placed in path d (Fig. 5.3). The state s_3 in turn samples an action a_{30} and the resulting optical state leads to a reward. Then, the weights of the edges $(1, 14)$ and $(3, 30)$ should be increased, to raise the probability to choose action a_{14} or a_{30} when the agent observes the states s_1 or s_3, respectively.

Let us formalize this idea and mathematically describe how the weights are updated. We denote the number of interaction circles (see Fig. 5.1) passed during training by t, and the weights after t cycles by $h_{i,j}^{(t)}$. The reward $r^{(t)} \in \{0, \lambda\}$ after t cycles is equal to $\lambda > 0$ if and only if the agent produced a state with desired SVR of $(3, 3, 2)$ or $(3, 3, 3)$. Then, the update rule for the weights used in [56] is

$$h^{(t+1)} = h^{(t)} + r^{(t)} g^{(t+1)}\,, \tag{5.11}$$

where $h^{(t)}$ is the weight matrix with entries $h^{(t)}_{i,j}$ and the *glow matrix* $g^{(t+1)}$ allows to redistribute rewards such that decisions that were made in the past are rewarded less than the more recent decisions. The glow matrix $g^{(0)} = \mathbb{O}$ is initialized with the zero matrix and is updated in each step t by setting the entry $g_{i,j} = 1$ if the edge (i, j) was traversed in the last decision process in the PS architecture. Furthermore, the g matrix is updated after each interaction with the environment to lower the extent to which actions far in the past are rewarded by

$$g^{(t+1)} = (1 - \eta)g^{(t)}, \tag{5.12}$$

where $\eta \in [0, 1]$ is the *glow parameter* of the PS model. The hyperparameter η can be chosen heuristically, or also learned by PS itself [106]. From (5.11) together with (5.12) one can then see that a positive reward $r^{(t)}$ increases the weights corresponding to recent transactions from states to actions.

5.2.2.2 Action Composition

Action composition [70] enables the PS agent with some primitive kind of creativity [102] by coming up with its own actions. In [56], a concrete version of this idea is implemented to learn actions that lead to a positive reward. Each time the PS agent achieves a positive reward, the sequence of actions that lead to this reward is added as an additional action clip to the PS model (Fig. 5.3). This allows the agent

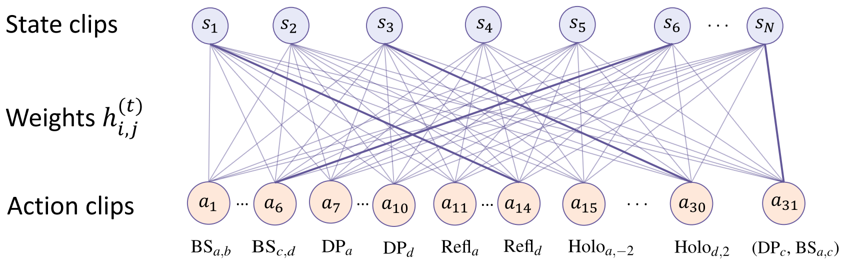


Fig. 5.3 Trained architecture for the PS agent [56]. The figure demonstrates how the architecture for the PS agent shown in Fig. 5.2 may have changed after t steps of training. State clips $s_2, \ldots, s_N$ that were explored during a rewarded experiment have been added during training. The probability that action a_j is chosen for an input state s_i is related to the weight $h^{(t)}_{i,j}$ by (5.13). The weight is increased if choosing action a_j in situation s_i leads to a positive reward in a situation experienced during training (high weights are depicted with solid lines in the figure). Hence, the probability to choose actions that will lead to rewarded experiments is increased during the training process. Furthermore, action composition changes the structure of the PS architecture. For example, the PS agent may realize during training that the combination of a Dove prism on path c together with a beam splitter on path a and c is particularly useful to achieve its goal. Hence, the composite action clip $(\mathrm{DP}_c, \mathrm{BS}_{a,c})$ is added, which allows the agent to place both elements in one training step

to directly place several optical tools that turned out to be useful in past experiences together in one step. Hence, the number of available actions varies over the number of training steps, and we denote this number after t training steps by $K^{(t)}$. The weights connecting the new action to all the state clips are initialized with value one and the probability to choose action a_j given state s_i is then given by

$$p_{i,j}^{(t)} = \frac{h_{i,j}^{(t)}}{\sum_{k=1}^{K^{(t)}} h_{i,k}^{(t)}} . \tag{5.13}$$

Rescaling of the tool box size $K^{(t)}$ leads to a smaller change of the probability $p_{i,j}^{(t)}$ with respect to a change of a weight. To compensate for this, we also rescale the reward $r^{(t)} \in \{0, \lambda\}$ by rescaling λ dependent on t proportionally to the size of the toolbox as

$$\lambda^{(t)} = \lambda^{(0)} \frac{K^{(t)}}{K^{(0)}} , \tag{5.14}$$

where $\lambda^{(0)}$ corresponds to the initial reward λ, and where a positive reward $r^{(t)} \in \{0, \lambda^{(t)}\}$ after t steps is rewarded with $\lambda^{(t)}$.

5.2.2.3 Clip Deletion

The number of all possible states of the optical setup grows with $\mathcal{O}\left(K^L\right)$, where k is the number of actions and L the maximal number of elements placed on the optical setup. Hence, adding all states that are built by the PS agent as a clip to the PS model may lead to a huge number of state clips, and hence, to a huge number of weights that must be trained. To avoid this, we delete state clips that are only used infrequently. More concretely, if, after placing a maximal number L of elements on the optical table, no positive reward was obtained, all state clips and edges that were created (and added to the PS architecture) are deleted.

Similarly, we also use a deletion mechanism to keep the total number of actions $K^{(t)}$ reasonable small. The idea is to only keep composite actions if they turn out to be useful in many situations. Technically, we check if after some training steps, the sum of the weights of the incoming edges to the added action clip is large enough to justify its existence. If this is not the case, we remove the action again. (For a full description of a method that implements this idea we refer to the Supplemental material of [56].)

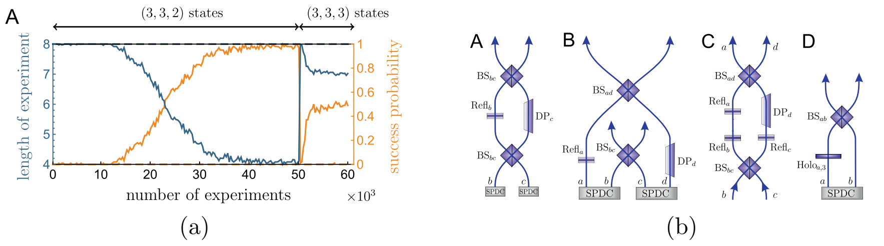


Fig. 5.4 Results for creating optical setups using PS (Figures taken from Melnikov and Nautrup et al., *Proceedings of the National Academy of Sciences* 2018 [56]). **a** The plot shows the evolution of the success probability as well as the average length (i.e., the number of optical elements) of the experiments during the training process. The PS agent is trained for 5×10^4 experiments with the goal of creating states with SRV (3, 3, 2), and subsequently for 10^4 experiments to create states with SRV (3, 3, 3). The trained agent successfully creates states with SRV (3, 3, 3) with probability of around 50%. The dashed lines corresponds to an agent that is trained with the goal of creating states with SRV (3, 3, 3) from the beginning. Such an agent does essentially not make any learning progress. This suggest that the techniques to create states with SRV (3, 3, 2) are also useful for the creation of states with SRV (3, 3, 3). **b** Experimental setups frequently used by the PS agent. (A) Local parity sorter. (B) Nonlocal parity sorter. (C) Nonlocal parity sorter in the Klyshko wave front picture [107], in which the paths a and d are identical to the paths b and c, respectively. (D) Setup to increase the dimensionality of the entanglement of photons

5.2.3 Results

The PS agent is trained for 5×10^4 experiments with the goal of learning photon states with SRV (3, 3, 2). The glow parameter is $\eta = 1/16$, the maximal number of optical elements is $L = 8$ and the initial size of the reward is $\lambda^{(0)} = 1$. The PS agent learns to create experimental setups that prepare photon states with SRV (3, 3, 2) with high probability and with a decreasing number of optical elements in the training process [56] (Fig. 5.4a). Then, the same PS agent is trained further with 10^4 experiments to learn creating photon states with SRV (3, 3, 3). Within this short additional period of training, the PS learns quickly to create such states with 50% success probability. This suggests that either knowing how to create states with SRV (3, 3, 2) helps the agent to construct states with SRV (3, 3, 3), or alternatively, that constructing states with SRV (3, 3, 3) is simpler than constructing states with SRV (3, 3, 2). To check this, a newly initialized PS agent is trained for 6×10^4 experiments with the goal of creating states with SRV (3, 3, 3), but the success probability stays near zero (Fig. 5.4a). This demonstrates that the agent trained to produce states with SRV (3, 3, 2) discovered some structures that also help in producing states with SRV (3, 3, 3). Some insight into how the PS agent builds the setups that lead to the desired states is discussed in Sect. 5.3.

5.3 Conceptual Insights from Action Composition

In Sect. 5.2, we discussed how PS can be used to create experimental setups for a specific task of preparing entangled states. As scientists, however, we are not only interested in knowing experimental setups that produce such states, but we also seek some understanding of the principles used to build the setups. Indeed, the connection between experimental setups and the entanglement structure of the created state is not well understood [55]. Action decomposition, which allows a PS agent to compose new actions during the training process that turned out to be particularly useful, can help to get some insight into this relation. Identifying gadgets often appearing in successful experimental setups can help to get some understanding of which building blocks are particularly useful for creating highly entangled states.

The added composed actions of the PS agent that was trained as described in Sect. 5.2 are analyzed in [56]. The importance of an action a_j is estimated according to the sum $\sum_k h_{k,j}$ of the weights of all incoming edges (which is related to the probability that the action a_j is chosen). One finds that the PS agent extensively uses a combination of actions corresponding to an optical interferometer (shown in Fig. 5.4b A). Usually, such interferometers are used to sort OAM modes with different parities [108], hence, the PS agent essentially (re-)discovered an optical setup that is well known and useful for various tasks.

To discover other relevant gadgets, one may train the PS agent in a way such that only novel experimental setups are rewarded. Hence, if the PS agent finds a setup that achieves an SRV (3, 3, 2) (or (3, 3, 3)), a reward is only given if this setup was not already found earlier in the training process. Analyzing again the composed actions found by such an agent, leads to several new interesting gadgets [56] (see Fig. 5.4b B,C, and D). The gadget shown in Fig. 5.4b B, a nonlocal interferometer, has recently been analyzed and motivated the work in [65]. Further, it can be shown that the gadget shown in Fig. 5.4b D allows to create high-dimensional entangled states beyond the initial state dimension of three in the OAM basis.

Chapter 6
Model Creation

In this chapter, we discuss how to optimize and build physical models using machine learning. There is an extensive literature on the use of machine learning to solve scientific (optimization) problems. Work in this direction has many success stories and could be used to solve many challenging problems in science. Often, a model for the physical system is assumed and the aim of the machine is mainly to optimize the parameters of the models. Therefore, despite the great practical usefulness of such approaches, they do not usually lead to new scientific knowledge beyond the specific solution to the given problem. In Sect. 6.1, we briefly review some recent work in this area. However, since the focus of this book is on discovering new models with machine learning, we keep this section short and provide several references for further reading. In Sect. 6.2, we discuss how to find mathematical formulas describing the time evolution of physical systems. Finding mathematical formulas is an essential part of a physicist's discovery process. The bias that the underlying model is described by "simple" mathematical formulas has led to models with great generalization power in history.

6.1 Optimizing Model Parameters

A lot of literature considers building machine learning systems modeling specific physical systems. Then, the parameters of the model are optimized to solve a specific problem. Such methods have proven very successful in solving challenging scientific problems, e.g. in the design of materials and molecules through accelerated simulations of quantum systems (see [109] for an extensive review using machine learning for quantum physics). Neural networks have become the most important machine learning tool for solving scientific problems in recent years. For example, in condensed matter physics and generally in many-body settings, neural networks have proven particularly useful to characterize phase transitions (see [14] and references

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_6

therein) and to learn local symmetries [110]. In quantum chemistry, neural networks and graph based networks show impressive results for synthesis of molecules and predicting their properties [111–113] (see [114, 115] for recent reviews and further references). A more extensive review on the applications of machine learning in physics can be found in [17].

The incorporation of prior knowledge about a particular physical model into machine learning systems fundamentally limits what can be "discovered" by the machine. In fact, we cannot claim a machine to discover a physical model itself if some knowledge about the model is already hardcoded in the machine learning system. Another approach, discussed in full detail in Part III of this book, is to model the physicist's reasoning process with the machine learning system. This approach has the potential to discover physical models on its own in tradeoff with the performance of solving specific physical problems. The tradeoff between the generality of a model and its performance is fundamental, in the sense that discovering something from scratch is always harder than starting from some prior knowledge about the system.

6.2 Discovering Physical Laws

Discovering the physical laws underlying experimental data is one of the main goals of a physicist. Automatizing this process was considered in an early study in [18] in 1987. In 2007 an algorithm was introduced in [116] that searches a space of mathematical expressions on given input variables to recover the differential equations underlying time-series data. Two years later, it was demonstrated in [19] that searching for conserved quantities in time-series data can be used to find the laws of motion of various mechanical systems, such as a double pendulum. In 2011, the same algorithm was applied to infer analytical models for metabolic networks [20]. In the last few years, significant progress was made in extracting dynamical equations from experimental data (see e.g. [117–120]), which is known to be an NP-hard problem [121]. The methods in [118], for example, have been successfully applied to complex physical systems such as water flowing through a pipe (defined by the Navier-Stokes equations). Recently, physical prior knowledge was incorporated into deep neural networks leading to so called *physics-imformed* deep neural networks [122]. Physics-informed neural networks have been used for the data-driven discovery of partial differential equations by exploiting the a priori knowledge that physical dynamics should belong to a class of partial differential equations [122–125]. Last but not least, tensor networks were used in [126] to learn non-linear dynamical laws. Building physical principles, such as the locality of interactions, into the tensor network naturally ensures scalability of the approach.

In the following sections, we will focus on the methods introduced in [19], because of three reasons: (i) the methods described in [19] cover the basic ideas for extracting mathematical laws from data, (ii) a minimal amount of prior knowledge on the expected form of the formulas is required, and (iii) the work in [19] provides a basis to understand more recent papers in this field. For readers interested in using the method that achieves the best performance in discovering physical laws from a given data set, please refer to the more recent references above.

All the work on extracting physical laws mentioned above assumes prior knowledge in the form of knowledge of the relevant variables describing the physical system. In Part III of this book we explain how to train a neural network to learn the relevant parameters and the differential equation describing the time evolution of the physical system in parallel (see Sects. 8.5, 9.6 and 10.3).

6.2.1 Symbolic Regression

Symbolic regression [21] is a form of regression in which the goal is to find a mathematical expression (the form as well as the parameters) that accurately describes the data. The accuracy and simplicity of the solution are equally essential. In the history of physics, the bias towards preferring "simple" mathematical models over complex formulas (which can be seen as a special case of Occam's razor, i.e., preferring simple explanations over complex ones) often lead to models with great generalization power. More concretely, given data samples (x, y) with $x \in \mathbb{R}^n$ for some $n \in \mathbb{N}$ and, without loss of generality, $y \in \mathbb{R}$, symbolic regression searches for a simple mathematical expression $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(x) \approx y$ for all samples (x, y). For example, given data points $(0, 0)$, $(1, 1)$, $(2, 4)$, we may expect to find formulas like $f(x) = x^2$. Of course, we could also fit the data with a third-degree polynomial, but we consider higher-degree polynomials to be more "complex" than lower-degree ones and therefore prefer expressions of low-degree polynomials. In praxis, the data samples are usually noisy and there is a tradeoff between accuracy and simplicity of the solutions.

The space of mathematical equations is infinitely large and it is a hard task to efficiently search for the simplest expression describing the samples in this space. Furthermore, to automatize the search of simple expressions, we have to define the "complexity" of an expression. Mostly, *genetic algorithms* [127] are used as a search method. Genetic programming is a machine learning technique that is often used for searching an element in a discrete space. In principle, it can also be applied to continuous spaces, however, often we have more efficient methods at hand in this case. The idea of genetic algorithms is based on evolutionary theory that attempts to solve a given task by developing potential solutions over generations, discarding the least fit contestants and mutating the fittest ones. This algorithm can theoretically be used to solve a wide range of problems, as long as a way to determine the fitness and to genetically modify a potential solution exists. In the following, we describe how to use genetic algorithms for symbolic regressions, hence where the *genotype* corresponds to encodings of symbolic expressions. The aim of the following discussion is to introduce the reader to the basic concepts, and not to provide all details required for an efficient implementation. For readers interested in implementing or using symbolic regression, we refer to [19] and to [128] for a discussion of symbolic regression with a focus for applications in physics.

Let us consider a set of samples $\mathcal{S} = \{(x^i, y^i)\}_i$, and represent mathematical expression with graphs where each node corresponds to a mathematical building block and each leave to a system variable or parameter (examples are shown in Fig. 6.1). Crucially, the choice of the set from which the mathematical building blocks are chosen, introduces a bias into the search over expressions. The genetic algorithm starts by randomly choosing a set $\mathcal{P}_0$, called *population*, of mathematical expressions using the system variables $(x_1, \ldots, x_n)$ corresponding to the coordinates of a sample $x^i \in \mathbb{R}^n$. The *fitness* $F(f, \mathcal{S})$ of an expression f may then be defined by the negative averaged mean square prediction error, i.e.,

$$F(f, \mathcal{S}) = -\frac{1}{|\mathcal{S}|} \sum_{i \in \{1,\ldots,|\mathcal{S}|\}} \left\| f(x^i) - y^i \right\|_2^2 . \tag{6.1}$$

Hence, the fitness is a measure of how well an expression approximates the given data samples. The complexity $C(f)$ of the expression f can be measured, for example, by the number of terms in the expression. To include the simplicity of the expression into the fitness, one may define

$$F'(f, \mathcal{S}) = -C(f) - \lambda \frac{1}{|\mathcal{S}|} \sum_{i \in \{1,\ldots,|\mathcal{S}|\}} \left\| f(x^i) - y^i \right\|_2^2 , \tag{6.2}$$

where the hyperparameter λ regulates the tradeoff between finding accurate and simple expressions.

Inspired by evolutionary theory, we calculate the fitness of each expression in the population $\mathcal{P}_0$ and further consider the expressions with the highest fitness. The fittest individuals are then "slightly" modified to generate a new population $\mathcal{P}_1$ (see Fig. 6.1 for an overview). Again motivated by evolutionary theory, the fittest expressions in the population are modified by *mutations* and *crossovers*. A biological mutation is a random change in the genes of an individual, and corresponds to replacing a small part of the graph representing the expression with a random graph in the considered case here (Fig. 6.1). On the other hand, crossover operations are analogous to the crossover that happens during sexual reproduction in biology. Here, they may be represented as cutting out a subgraph of a representation of an expression (with high fitness) and insert it into another expression (Fig. 6.1). The process of choosing the fittest individuals and generating a new population is then repeated $m \in \mathbb{N}$ times until a satisfying expression is found in the mth population $\mathcal{P}_m$.

Based on evolutionary theory, genetic search algorithms are expected to be more efficient than brute-force search provided that parts of fit candidates can be combined and mutated in a reasonable way, i.e., in a way such that combining two parts of fit individuals likely leads to another fit or maybe even fitter individual. Nontheless, the runtime of such algorithms often prohibits searching for complex expressions in many variables. Indeed, in [19], an exponential scaling in the complexity of the target

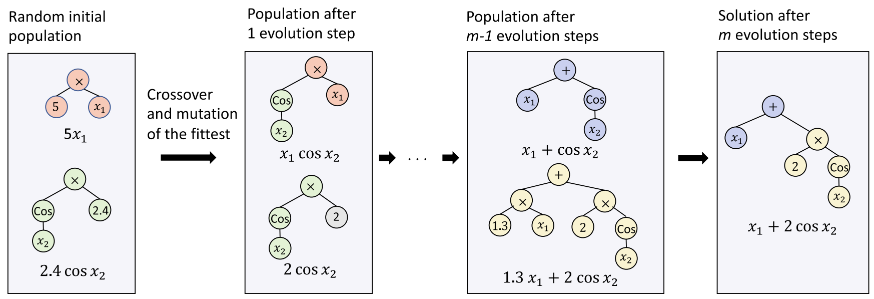

Random initial population
5
x_1
$5x_1$
Cos
x_2
2.4
$2.4\cos x_2$
Crossover and mutation of the fittest
Population after 1 evolution step
$x_1 \cos x_2$
2
$2\cos x_2$
Population after m-1 evolution steps
$x_1 + \cos x_2$
1.3
$1.3\,x_1 + 2\cos x_2$
Solution after m evolution steps
$x_1 + 2\cos x_2$

Fig. 6.1 Symbolic regression. The goal of symbolic regression is to find a simple mathematical expression f fitting with a given set $\mathcal{S} = \{(x^i, y^i)\}_i$ of input-output pairs, i.e., we search for a simple symbolic expression with $f(x^i) \approx y^i$ for all $i \in 1, \ldots, |\mathcal{S}|$ with $x^i = (x_1^i, \ldots, x_n^i) \in \mathbb{R}^n$ and $y \in \mathbb{R}$. For concreteness, here we consider $n = 2$ and a data set with $y^i \approx x_1^i + 2\cos x_2^i$, i.e., the target expression we would like to find is $f(x) = x_1 + 2\cos x_2$. Often, genetic algorithms are used to find the fittest expression, i.e., the simplest expression that fits the data well. The genetic algorithm starts from a random initial set of several symbolic expressions represented as graphs in the system variables x_1 and x_2 (the number of expressions depicted in the figure is not representative). Then, the fittest expressions are selected and changed by mutations and crossover operations. The upper graph in the initial population in the figure is combined with the lower graph by a cross over operation, corresponding to combining genes during sexual reproduction in biology. The lower graph is changed by a mutation of the leaf storing the parameter 2.4 by replacing it with a random number. We repeat this procedure $m \in \mathbb{N}$ times, until we end up with a simple expression fitting the data

expression was observed. A promising approach to lower the runtime by creating symbolic expressions in a step-wise fashion was proposed in [129] (see Remark 6.1).

Remark 6.1 (Stepwise symbolic regression) In [129], an approach is suggested to find symbolic expressions by step-wise adding system variables. The approach has similarities to the one in [19] and has not been fully automatized yet, but its idea is pretty simple and explained on the basis of an example in the following. Let us consider a sample set with pairs $((x, y, z), f(x, y, z))$ with an expression $f = z\ \sin(xe^y)$. The goal is to find the symbolic expression from the sample set. Assuming that we have numerical access to partial derivatives, we may numerically create samples for the following quotient

$$q_{x,y} = \frac{\partial f/\partial x}{\partial f/\partial y} = \frac{z\cos(xe^y)e^y}{z\cos(xe^y)xe^y} = \frac{1}{x}. \tag{6.3}$$

We may use (standard) symbolic regression to obtain an expression $f_{x,y}$ for the samples $((x, y), q_{x,y}(x, y))$ such that the quotient $\frac{\partial f_{x,y}/\partial x}{\partial f_{x,y}/\partial y}$ approximates the numerical values of $q_{x,y}(x, y)$. A simple expression that symbolic regression may find is $f_{x,y} = xe^y$, hence we recovered the relation between two of the system variables in the target expression. The form of the target expression is then expected to be given as $f = g(xe^y, z)$ for some function $g : \mathbb{R}^2 \to \mathbb{R}$.

In a second step, we may consider the relation between x and z by investigating (for a fixed y with $e^y = c \in \mathbb{R}$)

$$q_{x,z} = \frac{\partial f/\partial x}{\partial f/\partial z} = \frac{cz\cos(cx)}{\sin(cx)}. \tag{6.4}$$

Again, we calculate the partial derivatives numerically to create samples of the quotient and apply symbolic regression to find an expression $f_{x,z} = z\ \sin(cx)$. Then, the target function is expected to be of the form $f = g'(z\sin(c(y)x), y)$ for some $g' : \mathbb{R}^2 \to \mathbb{R}$ and $c : \mathbb{R} \to \mathbb{R}$. From the requirement $f = g(xe^y, z)$, we see that setting $g(a, z) = z\sin(a)$ provides the same expression as given by $f = g'(z\sin(c(y)x), y)$ setting $g'(a, y) = a$ and $c(y) = e^y$. Hence, we discovered the target expression $f = z\ \sin(xe^y)$ by applying twice symbolic regression to find simpler expressions in only two system variables, which can be done more efficiently than directly searching for the full expression. However, combining the expression may not always be easy and has not been automatized so far. It would be interesting to consider this for future work.

6.2.2 Extracting Physical Laws from Data

In this section, we describe the ideas used in [19] to discover physical laws from time-series data. The approach in [19] was successfully applied to find the equation of motion of various physical systems including the chaotic double-pendula without using any prior knowledge about physics, apart from the knowledge of the system variables (discovering the system variables with deep neural networks is discussed in Part III of this book). The conceptual basis of the approach in [19] is that nearly all physical laws in nature are based on mathematical symmetries and invariants [130], implying that the search for many natural laws is inextricably linked to the search for conserved quantities and invariant equations [131, 132]. The main difficulty in searching for conserved quantities using symbolic regression is to avoid finding trivial invariants, such as $\cos^2(x) + \sin^2(x)$ for a system variable $x \in \mathbb{R}$, which is constantly equal to one but does not contain any physical insights. In the following, we explain the approach taken in [19] to avoid such trivial invariants on the basis of an example.

Let us consider a pendulum, i.e. a non-linear oscillator whose position is described by an angle θ. The sample set $\mathcal{S} = \{(t^i, \theta(t^i), \omega(t^i))\}_i$ consists of triples containing the time t^i, as well as the corresponding position $\theta(t^i)$ and angular velocity $\omega(t^i)$. Note that we have no values provided for the conserved quantities we are searching for, hence, they must be learned in an unsupervised way. The secret to computationally detecting nontrivial conservation laws is that the candidate equations should predict relations between the dynamics of the system's subcomponents. More specifically, the conservation equation should be able to predict relationships between derivatives of groups of variables over time. Such relations can be captured by the quotient of partial derivatives (in a similar spirit as we discussed in Remark 6.1). In other words, the expression $f(\theta, \omega)$ for a conserved quantity is considered to be non-trivial if it satisfies

$$\frac{\partial\theta/\partial t}{\partial\omega/\partial t} = \frac{\partial\theta}{\partial\omega} = \frac{\partial f/\partial\omega}{\partial f/\partial\theta} . \tag{6.5}$$

We can numerically estimating the left hand side of (6.5) as

$$\frac{\partial\theta/\partial t}{\partial\omega/\partial t} \approx \frac{\Delta\theta}{\Delta\omega} , \tag{6.6}$$

where $\Delta\theta$ and $\Delta\omega$ denote the changes of the parameters θ and ω within a small time interval Δt. The fitness of an expression f can then be estimated by the *mean-log-error* [19]

$$F(f, \mathcal{S}) = -\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \log\left(1 + \left|\frac{\Delta\theta(t^i)}{\Delta\omega(t^i)} - \frac{\partial f/\partial\omega}{\partial f/\partial\theta}\bigg|_{t^i}\right|\right) , \tag{6.7}$$

where $\Delta\theta(t^i)$ and $\Delta\omega(t^i)$ estimate the time derivative of the variables θ and ω at time t^i (using the set $\mathcal{S}$) and where the partial derivative is calculated symbolically using the analytical expression for the candidate function f.

We can then use symbolic regression (as described in Sect. 6.2.1) to search for candidate functions f with the highest fitness (see Fig. 6.2 for an overview).[1] In [19], it was demonstrated, that this method recovers the invariant expression $f(\theta, \omega) = c_1\omega^2 + c_2\cos(\theta)$ for two parameters $c_1, c_2 > 0$ depending on the mass m and the length l of the pendulum.

The invariant expression corresponds to the Lagrangian of the pendulum, however we have left to find the dependence of the parameters c_1 and c_2 on the mass and length of the pendulum. Since the invariant f is scale invariant, we may divide it by c_2 leading to $\tilde{f}(\theta, \omega) = c_1/c_2\omega^2 + \cos(\theta)$ and search for a symbolic expression of c_1/c_2 in the system variables m and l. The expression $c_1/c_2 \equiv k(m, l)$ can for example be found as follows: One creates time-series data for pendulums with different masses m_j and of different length l_j. For each pendulum j, we determine the value of k_j such that the expression $k\omega^2 + \cos(\theta)$ optimally fits the time-series data, in the sense that it stays approximately constant. Then, we can again run symbolic regression with the system variables m and l and the sample set $\{([m_j, l_j], k_j)\}_j$, i.e., we are

[1] For simplicity, we do not go into the technical details here of how to manage the tradeoff between high fitness and low model complexity. One way to solve this problem is presented in [19]. An alternative method, presented in [133], uses a corpus of closed-form mathematical models from Wikipedia to obtain the posterior probability of each expression from basic probabilistic arguments and explicit approximations. The posterior probability leads to a prior over expressions that accounts for model complexity.

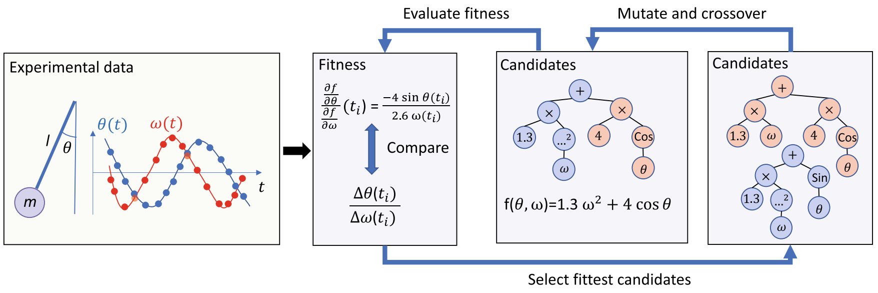

Evaluate fitness
Mutate and crossover
Experimental data
l
θ
m
$\theta(t)$ $\omega(t)$
t
Fitness
$\frac{\partial f / \partial \theta}{\partial f / \partial \omega}(t_i) = \frac{-4 \sin\theta(t_i)}{2.6\,\omega(t_i)}$
Compare
$\frac{\Delta\theta(t_i)}{\Delta\omega(t_i)}$
Candidates
$f(\theta,\omega)=1.3\,\omega^2 + 4\cos\theta$
Candidates
Select fittest candidates

Fig. 6.2 Extracting invariant expressions from time-series data [19]. We consider experimental data collected from a pendulum whose position at a time t is described by an angle $\theta(t)$ and we denote the angular velocity by $\omega(t) := \dot{\theta}(t)$. Symbolic regression is used to extract time-invariant expressions, i.e., conserved quantities, in θ and ω from a data set $\mathcal{S} = \{(t^i, \theta(t^i), \omega(t^i))\}_i$ consisting of time series of the angle and velocity of the pendulum. Starting from randomly generated symbolic expressions (represented as graphs), we evaluate their fitness according to (6.7). The chosen expression (6.7) for the fitness is a measure of how well a candidate expression is able to predict relations between the dynamics of the system's subcomponents. Hence, a high fitness points towards an invariant expression that contains non-trivial physical information. Then, the fittest candidate expressions are chosen, and modified by mutations and crossover operations with the aim of creating even fitter individuals. This process is repeated until an invariant expression with fitness approximately equal to zero is found, which corresponds to an expression that can approximately recover the relation between the dynamics of the angle $\theta(t)$ and the velocity $\omega(t)$. In the considered example, the found expression is the Lagrangian of the pendulum given by $f(\theta, \omega) = c_1\omega^2 + c_2 \cos(\theta)$ for two parameters $c_1, c_2 > 0$ depending on the mass m and the length l of the pendulum

searching symbolic expressions $k(m, l)$ with $k(m_j, l_j) \approx k_j$ for all j.[2] Finally, we find the form of the Lagrangian to be independent of the mass given by $L(\theta, \omega) = l/(2g)\,\omega^2 + \cos(\theta)$ with $g = 9.81\,m/s^2$ being the gravitational acceleration on the surface of Earth.

The equation of motion is then recovered by the Euler-Lagrange equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}L}{\mathrm{d}\omega}\right) - \frac{\mathrm{d}L}{\mathrm{d}\theta} = \frac{l}{g}\ddot{\theta} + \sin\theta = 0\,, \tag{6.8}$$

and hence $\ddot{\theta} = -g/l\,\sin(\theta)$. Alternatively, we could search for invariant expressions in the system variables θ, ω and α, where $\alpha := \dot{\omega}$ denotes the acceleration of the pendulum. Doing so, we would directly recover the equation of motion [19]. This demonstrates on one hand the dependence of the approach on the chosen system variables, and on the other hand also that we discover useful invariants in settings using different system variables.

The runtime of the genetic search algorithm for finding invariant expressions takes from some minutes up to 30 h for the double pendulum parallelizing the search by

[2] Here we use the knowledge that the pendulum is fully described by its mass and length. Recovering these parameters automatically is discussed in Part III of this book.

using 32 cores [19]. On the other hand, finding the expressions for the parameters usually takes significantly less time. We conclude that when considering systems with only a few known system variables, the described approach provides a powerful method to find symbolic expressions for invariant quantities. The symbolic nature of the result can help a physicist to gain new conceptual insights into the system under consideration. On the other hand, the prior assumption that simple analytic expressions exist may prohibit us from recovering invariants that cannot be described as simple analytic formulas in the chosen system variables. For systems containing of many (probably unknown) system variables one way consider more recent methods in this field (see the introduction of Sect. 6.2 for relevant references) together with the methods introduced in Part III of this book.

Chapter 7
Model Testing

Testing physical models is an essential part of a physicist's discovery process (discussed in Sect. 1.2). Falsifying a model in a certain physical domain (such as in a high energy range) helps to understand the limits of a model and guides future research towards improved models with a broader application range. Testing new models is of particular interest in high-energy physics. Although the Standard Model (SM) of particle physics very successfully describes a large number of elementary particle processes with high accuracy, unexplained phenomena such as the neutrino masses or the appearance of dark matter in the universe point to the need to extend or adapt the SM. Searching for new phenomena in high-energy physics often relies on testing suggestions of new models by running experiments (e.g. with the Large Hadron Collider) and then comparing the collected data with predictions of a *reference model* (e.g. the SM of particle physics). Often, the considered models make probabilistic predictions, which makes testing particularly challenging. In several papers, a reference model was tested against another hypothetical model by constructing hypothesis tests using standard techniques (see [134] for a review for model testing in particle physics). Such an approach requires to come up with an alternative model and has the disadvantage that a statistical test designed for a particular alternative hypothesis is typically insensitive to data deviations of other kinds from the reference model.

How can we discover the limitations of a model from test data without having an alternative model at hand? The basic idea is to test the reference model against a model constructed by a machine based on the (experimental) test data. If the class of models is not restricted to a specific class of physical models, we call this a *model-independent* strategy (in a physical sense). If the machine constructs a model that deviates significantly from the reference model, one can conclude that the reference model does not fit the test data well. Furthermore, we can analyze the regions where the two models diverge the most, which can lead to conceptual insights into what problems the reference model has. Such model-independent searches for new physics are less commonly considered in the literature [135–139, 139–142] than model-dependent searches. The methods described in [141, 142] follow similar ideas

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_7

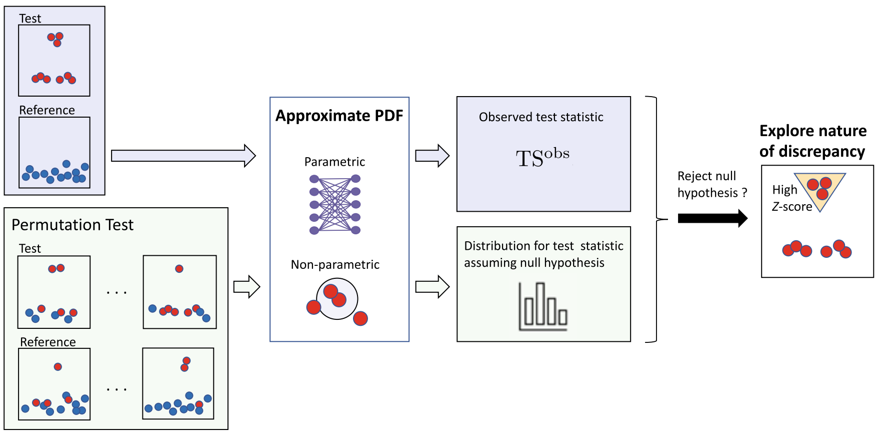


Fig. 7.1 Overview of checking model compatibility [141, 142]. The figure shows how to test the compatibility of a set $\mathcal{T}$ of test samples with a set of reference samples $\mathcal{R}$ in a model-independent way. The samples depicted in the figure live in a $d = 2$ dimensional feature space. The null hypothesis is that both data sets $\mathcal{T}$ and $\mathcal{R}$ are generated by the same model, i.e., the same probability density function (PDF). First, the test and the reference samples are provided as an input to a machine learning system whose goal is to estimate the PDFs underlying the test and the reference set and to provide the observed test statistic $\mathrm{TS}^{\mathrm{obs}}$ as an output (scenario marked in blue in the figure). The PDFs can be estimated with a parametrized model (such as neural networks [141]) or a non-parametrized method (such as the k-nearest neighbor estimation [142]). Here we use the test statistic given in (7.5), which approximates the KL-divergence between the true underlying test and reference PDF. To interpret the observed value $\mathrm{TS}^{\mathrm{obs}}$, we compare it with the probability distribution of the test statistic assuming that the null hypothesis is true. The distribution is obtained by the permutation test, i.e., by shuffling samples between the test and reference set and evaluating the test statistic for the obtained test and reference sets (scenario marked in green). Then, we can estimate the p-value, i.e., the probability that we observe $\mathrm{TS}^{\mathrm{obs}}$ or more extreme values of the test statistic assuming the null hypothesis. If the p value is lower than the desired significance level $\alpha \in [0, 1]$ of the test, the null hypothesis is rejected. Finally, we use the Z-score given in (7.16) to locate regions of high incompatibility in the feature space

for model-independent testing of a reference model. These ideas are described in the following (and summarized by Fig. 7.1) and can in principle be applied to test any physical model and to find regions where the collected data deviates significantly from the reference model. However, applying such techniques is praxis is still challenging due to reasons discussed in Sect. 7.5.

7.1 Statistical Setting

In this section, we describe the formal statistical setting for model-independent testing (following the lines in [142]). The setting considers a basic scenario of comparing two data sets with each other and can be applied to statistical problems beyond model testing. We consider two data sets with d-dimensional real data points

$$\mathcal{T} = \{x_i : x_i \in \mathbb{R}^d\,, 1 \leq i \leq n_T\} \quad \text{with } x_i \sim P_T \text{ and} \tag{7.1}$$
$$\mathcal{R} = \{x_i' : x_i' \in \mathbb{R}^d\,, 1 \leq i \leq n_R\} \quad \text{with } x_i \sim P_R\,, \tag{7.2}$$

where the observations $x_i \in \mathbb{R}^d$ and $x_i' \in \mathbb{R}^d$ are sampled i.i.d. from a test and a reference probability density function (PDF) P_T and P_R, respectively. The d-dimensional space in which the samples live is called the *feature space*. For the following considerations, neither P_T nor P_R have to be known analytically. In a typical practical scenario, the reference set $\mathcal{R}$ is created by running a simulation on a computer based on the reference model, while the test set $\mathcal{T}$ is obtained experimentally.

The goal is to determine wether the two data sets are (in)compatible, i.e., in a language of hypothesis testing, we would like to find out up to which significance level the null hypothesis $\{H_0 : P_T = P_R\}$ is rejected. A well-defined statistical test scenario requires an alternative hypothesis, which we define as follows $\{H_1 : P_T = \hat{P}_{\mathcal{T}}\}$, where the PDF $\hat{P}_{\mathcal{T}}$ depends on the test data set $\mathcal{T}$ and is chosen to approximate the true PDF P_T as accurately as possible. The choice of approximation method leads to different alternative hypotheses, whose choice is crucial for the sensitivity of the test. The goal considered here is to find approximation methods that lead to tests that are sensitive to a broad range of data deviations from the reference model. This is a subtle problem since every unjustified assumption on the class of PDFs used to approximate P_T may cause insensitivity to the data deviations occurring in the test sample.

As a measure of compatibility, we may consider the ratio λ of probability densities to observe the test data $\mathcal{T}$ under the PDF P_T (numerator) and under the PDF P_R (denominator) [142]

$$\lambda = \prod_{x_i \in \mathcal{T}} \frac{P_T(x_i)}{P_R(x_i)}\,. \tag{7.3}$$

A value of $\lambda \gg 1$ means that the data in the test sample $\mathcal{T}$ is much more likely to be sampled from the PDF P_T than from P_R. Since the PDFs are not known, we have to approximate them by $\hat{P}_{\mathcal{T}} \approx P_T$ and $\hat{P}_{\mathcal{R}} \approx P_R$ (see Sect. 7.2 for approximation strategies) to find an estimate on λ as

$$\hat{\lambda} = \prod_{x_i \in \mathcal{T}} \frac{\hat{P}_{\mathcal{T}}(x_i)}{\hat{P}_{\mathcal{R}}(x_i)}\,. \tag{7.4}$$

As a test statistic $\mathrm{TS}(\mathcal{T}, \mathcal{R})$, we may use [142]

$$\mathrm{TS}(\mathcal{T}, \mathcal{R}) = \log \hat{\lambda}^{\frac{1}{n_T}} = \frac{1}{n_T} \sum_{i=1}^{n_T} \log \frac{\hat{P}_{\mathcal{T}}(x_i)}{\hat{P}_{\mathcal{R}}(x_i)} \, . \tag{7.5}$$

The test statistic is used as a distance measure between the two PDFs $\hat{P}_{\mathcal{T}}$ and $\hat{P}_{\mathcal{R}}$ (Remark 7.1). A value of TS close to zero means that the two PDFs are essentially the same and hence, we may not reject the null hypothesis. On the other hand, a large value of TS suggests that the PDF $\hat{P}_{\mathcal{T}}$ underlying the test set is significantly different from the PDF $\hat{P}_{\mathcal{R}}$ underlying the reference set, and hence that we should reject the null hypothesis. How to determine the threshold value of TS above which we reject the null hypothesis for a desired test significance $\alpha \in [0, 1]$ is discussed in Sect. 7.3. First, we discuss how we can approximate the PDF with methods from machine learning in Sect. 7.2.

Remark 7.1 (Test statistic as an approximation of the Kullback-Leibler divergence [142]) The Kullback-Leibler (KL) divergence (see also Definition 8.4) of the PDFs P_T and P_R is defined as

$$D_{\mathrm{KL}}\left(P_T \| P_R\right) = \int P_T(x) \log \left(\frac{P_T(x)}{P_R(x)} \right) \mathrm{d}x \, , \tag{7.6}$$

where we integrate over the full feature space. The KL divergence is a fundamental measure in information theory that can be seen as a distance measure between two PDFs. Indeed, it is non negative and it equals zero if and only if $P_T = P_R$ (however, it is not a metric, since it is not symmetric). Replacing the integral with an empirical average over the samples in the test set, and P_T and P_R with the approximations $\hat{P}_{\mathcal{T}}$ and $\hat{P}_{\mathcal{R}}$, respectively, we see that the expression of the test statistic given in (7.5) approximates $D_{\mathrm{KL}}\left(P_T \| P_R\right)$.

7.2 Approximation of Probability Density Functions

In this section, we discuss how to obtain an estimate $\hat{P}_{\mathcal{R}}$ (or $\hat{P}_{\mathcal{T}}$) for the PDF P_R (or P_T) underlying a sample set $\mathcal{R}$ (or $\mathcal{T}$). More specifically, since we are ultimately interested in calculating the test statistic given in (7.5), we only need to estimate the quotient of the PDFs P_T and P_R at any point in the test set (a comprehensive review on density ratio estimators can be found in [143]). One can distinguish two types of approaches [141, 142]:

1. **Parametric modelling**: We consider a parametrized class of PDFs $\mathcal{C}_\Theta = \{\hat{P}_\theta\}_{\theta\in\Theta}$ from which $\hat{P}_{\mathcal{R}}$ is chosen from.
2. **Non-parametric modelling**: We do not restrict the class of PDFs and construct $\hat{P}_{\mathcal{R}}$ directly from $\mathcal{R}$.

The following sections present the general ideas behind both approaches on the basis of a specific machine learning strategy for leaning PDFs. Which method should be chosen to achieve the highest sensitive to new physics signals depends on the specific setting including the reference model and what kind of new signals are expected. For most methods, we have no exact understanding about the class of signals that can be detected with it with high significance. Hence, the choice of the model also depends on some experience of the user with various methods. The goal of the following sections is not to discuss many examples to gain such experience, but rather to introduce the reader to the general idea of how to approximate PDFs to search for unknown physical signals in data.

7.2.1 Parametric Modelling

In the parametric approach, first one has to choose the class of PDFs $\mathcal{C}_\Theta$. Instead of choosing a parametrized class of physical models, here we are interested in choosing an expressiv class that can approximate a wide range of PDFs. One option is to parametrize the class of PDFs by neural network [141], which are low-biased and universal function approximators (see Sect. 3.4). Furthermore, neural networks represent smooth functions, which is usually a reasonable bias for PDFs coming from physical models. Indeed, we would not expect a reasonable physical model to predict a discontinuous PDF. Last but not least, there is evidence that neural networks can break the curse of dimensionality. The number of events to approximate a PDF with a histogram grows exponentially with the dimension d of the feature space. While there is still no proof, the work in [144–146] suggests that neural network require fewer events to approximate PDFs on high-dimensional feature spaces. Since in practical applications, the dimension d is usually quite large, this is an extremely desirable property.

Having chosen a class of PDFs $\mathcal{C}_\Theta = \{\hat{P}_\theta\}_{\theta\in\Theta}$ (were the parameter θ contains all the weights and biases of the neural network), we can use the maximum likelihood method to approximate the PDFs $P_\mathcal{T}$ and P_R, i.e.,

$$\hat{P}_\mathcal{T} = \hat{P}_{\theta_\mathcal{T}} \text{ with } \theta_\mathcal{T} = \underset{\theta\in\Theta}{\arg\max} \prod_{x_i\in\mathcal{T}} \hat{P}_\theta(x_i) \text{ and} \tag{7.7}$$

$$\hat{P}_\mathcal{R} = \hat{P}_{\theta_\mathcal{R}} \text{ with } \theta_\mathcal{R} = \underset{\theta\in\Theta}{\arg\max} \prod_{x_i'\in\mathcal{R}} \hat{P}_\theta(x_i')\,. \tag{7.8}$$

From these approximations, we can calculate the test statistic $\text{TS}(\mathcal{T}, \mathcal{R})$ given in (7.5). Instead of using two neural networks, and solving both optimization problems given in (7.7) and (7.8), one can also directly represent the (logarithm of the) quotient of the PDFs with a neural network and minimize the test statistic. This method is described in full detail in [141] and is demonstrated to have high sensitivity to various data deviations.

7.2.2 Non-parametric Modelling

In the non-parametric approach, the PDFs are approximated without restricting to a parametrized model class. Here, we discuss the *nearest neighbors* (NN) approach [147–153], described in the context of model-independent testing in [142]. The basic idea is to estimate a PDF at a point x in the feature space by calculating the volume of the sphere that contains the $k \in \mathbb{N}$ nearest neighbors of x. The density estimate is then proportional to the number of points k divided by the volume of the sphere. Let us formalize this on the basis of estimating the probability density $\hat{P}_{\mathcal{T}}$ on the test set $\mathcal{T}$. Consider a sample $x_i \in \mathcal{T}$ and calculate the euclidean distance to all other samples in $\mathcal{T} \setminus \{x_i\}$. Then, the k-nearest neighbors (i.e., the k samples in $\mathcal{T} \setminus \{x_i\}$ with the smallest distances to x_i) are determined, and the radius $r_{i,\mathcal{T}}$ is set equal to the radius of the kth nearest neighbor. Hence, the k nearest neighbors of x_i are contained in a sphere of volume $\omega_d r_{i,\mathcal{T}}^d$ with centre at x_i, where ω_d denotes the volume of the d-dimensional unit sphere, i.e.,

$$\omega_d = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2}+1\right)}, \tag{7.9}$$

where Γ denotes the Gamma function, which is given by $\Gamma(n) = (n-1)!$ for positive integers n. Hence, normalizing with the total number of samples $(n_T - 1)$ in $\mathcal{T} \setminus \{x_i\}$, the NN-estimation of the PDF P_T is given by

$$\hat{P}_{\mathcal{T}}(x_i) = \frac{k}{(n_T - 1)} \frac{1}{\omega_d\, r_{i,\mathcal{T}}^d}, \tag{7.10}$$

for any $x_i \in \mathcal{T}$. To estimate the reference PDF P_R on samples $x_i \in \mathcal{T}$ in the test set, we calculate the distances from x_i to the k nearest neighbors in $\mathcal{R}$. The distance to the kth nearest neighbors is denoted by $r_{i,\mathcal{R}}$, and the NN-estimation of the reference PDF P_R is then given by

$$\hat{P}_{\mathcal{R}}(x_i) = \frac{k}{n_R} \frac{1}{\omega_d\, r_{i,\mathcal{R}}^d}, \tag{7.11}$$

for any $x_i \in \mathcal{T}$. The test statistic $\mathrm{TS}(\mathcal{T}, \mathcal{R})$ given in (7.5) then simply becomes

$$\mathrm{TS}(\mathcal{T}, \mathcal{R}) = \log \frac{n_R}{n_T - 1} + \frac{d}{n_T} \sum_{i=1}^{n_T} \log \frac{r_{i,\mathcal{R}}}{r_{i,\mathcal{T}}}. \tag{7.12}$$

It can be shown that the expression in (7.12) is a consistent and asymptotically unbiased estimator of the KL-divergence [154–156], i.e., the test statistic (7.12) converges almost surely to the KL-divergence $D_{\mathrm{KL}}(P_{\mathcal{T}}, P_{\mathcal{R}})$ between the true PDFs $P_{\mathcal{T}}$ and $P_{\mathcal{R}}$ in the large sample limit $n_{\mathcal{R}}, n_{\mathcal{T}} \to \infty$.

7.3 Statistical Hypothesis Testing

The ultimative goal of the statistical setting described in Sect. 7.1 is to decide wether the null hypothesis $\{H_0 : P_{\mathcal{T}} = P_{\mathcal{R}}\}$ can be rejected with a significance level $\alpha \in [0, 1]$. For example, rejecting the null hypothesis with significance $\alpha = 0.05$ means that the probability to observe the test data assuming the null hypothesis is below 5%. To interpret the observed value of the test statistic $\mathrm{TS}(\mathcal{T}, \mathcal{R})$ given in (7.5), we have to derive a probability distribution $P_{\mathrm{TS}|H_0}$ over the values of TS under the assumption that the null hypothesis is true, i.e., assuming that $P_{\mathcal{T}} = P_{\mathcal{R}}$. Then, we can compute the two-sided p value of the null hypothesis, i.e., the probability that the value of the test statistic is at least as far away from the expected value under the null hypothesis as the observed test statistic $\mathrm{TS}^{\mathrm{obs}}$.

As suggested in [142], we us a resampling method known as the permutation test [157, 158] to compute the distribution $P_{\mathrm{TS}|H_0}$. The basic idea is that we resample various test and reference sets under the null hypothesis that both sets are generated by the same PDF (see also Fig. 7.1). Then, we evaluate the test statistic for the resampled sets and finally, we derive a distribution over the test statistic from the various data points that we obtained.

More formally, first we merge the test and reference set into a pool of samples $\mathcal{U} = \mathcal{T} \cup \mathcal{R}$. Then we randomly choose $n_{\mathcal{T}}$ samples from $\mathcal{U}$ and assign them to a set $\tilde{\mathcal{T}}$.[1] The remaining elements are assigned to a resampled reference set $\tilde{\mathcal{R}}$. Then, the test statistic for the set $\tilde{\mathcal{T}}$ and $\tilde{\mathcal{R}}$ is calculated. Repeating this procedure n_{perm} times, leads to a set $\{\mathrm{TS}_1, \ldots, \mathrm{TS}_{n_{\mathrm{perm}}}\}$ of n_{perm} data points for the test statistic. Dependent on the size of n_{perm}, we may choose an interval size $\Delta\mathrm{TS}$ and discretize the distribution $P_{\mathrm{TS}|H_0}$ with a histogram by assigning the data points TS_j to the bins $[m\Delta\mathrm{TS}, (m + 1)\Delta\mathrm{TS}]$ for $m \in \mathbb{Z}$. To decide wether the null hypothesis should be rejected with significance level α, we calculate the p-value defined by

$$p = \int_{-\infty}^{\hat{\mu}-\delta} P_{\mathrm{TS}|H_0}(t)\mathrm{d}t + \int_{\hat{\mu}+\delta}^{\infty} P_{\mathrm{TS}|H_0}(t)\mathrm{d}t\,, \tag{7.13}$$

[1] Alternatively, one may think of the elements in $\mathcal{U}$ to have a fixed order and then applying a random permutation on the list of elements (which is the reason for the name "permutation test"). Assigning the first $n_{\mathcal{T}}$ elements to $\tilde{\mathcal{T}}$ leads to the same result as randomly sampling these elements from the set $\mathcal{U}$.

where $\hat{\mu}$ denotes the estimated mean value of the test statistic, $\delta = |\mathrm{TS}^{\mathrm{obs}} - \hat{\mu}|$ and where the integration is actually implemented as a summation over the histogram that approximates the distribution $P_{\mathrm{TS}|H_0}$. Finally, the null hypothesis is rejected if the p value is smaller than the significance level α, since in such a case, the probability that we observe a value of the test statistic that deviates at least by δ of the expected value $\hat{\mu}$ is smaller than α assuming the null hypothesis.

Summarizing, the statistical procedure is as follows:

1. Approximate the distributions P_T and P_R with $\hat{P}_{\mathcal{T}}$ and $\hat{P}_{\mathcal{R}}$ (at data points $x_i \in \mathcal{T}$).
2. Calculate the observed test statistic $\mathrm{TS}^{\mathrm{obs}}(\mathcal{T}, \mathcal{R})$ using (7.5).
3. Estimate the distribution $P_{\mathrm{TS}|H_0}$ over the test statistic under the assumption that the null hypothesis is true using the permutation test.
4. Compute the two-sided p value of the null hypothesis.
5. If the two-sided p value is smaller than α, we reject the null hypothesis.

7.4 Identifying the Discrepant Regions

Let us assume that the statistical test described in Sect. 7.3 leads to a rejection of the null hypothesis. This means that the observed test statistic $\mathrm{TS}^{\mathrm{obs}}$ obtains a value that is far away from the expected value under the null hypothesis. To get some conceptual insight into why the reference model can not accurately explain the test data, we investigate the region in the feature space that contributes significantly to the deviation of the test statistic from the expected value. To find such regions, we follow the approaches in [141, 142] and assign a density field $U(x_i)$ to each sample $x_i \in \mathcal{T}$, given by

$$U(x_i) \equiv \log \frac{\hat{P}_{\mathcal{T}}(x_i)}{\hat{P}_{\mathcal{R}}(x_i)} \,. \tag{7.14}$$

The observed test statistic (7.5) can then be considered as the empirical average over the density field representing the expectation value

$$\mathrm{TS}^{\mathrm{obs}} = \mathbb{E}_{\mathcal{T}}[U] \,. \tag{7.15}$$

Furthermore, we normalize the density field to obtain a *Z-score*

$$Z(x_i) \equiv \frac{U(x_i) - \mathbb{E}_{\mathcal{T}}[U]}{\sqrt{\mathbb{V}_{\mathcal{T}}[U]}} \,, \tag{7.16}$$

where $\mathbb{V}_{\mathcal{T}}[U]$ denotes the variance of the density field U. To simplify the following discussion, let us assume that $\text{TS}^{\text{obs}} > \hat{\mu}$, where $\hat{\mu}$ is the expected value for the test distribution under the null hypothesis. The Z-score can be used to identify samples in $\mathcal{T}$ with a significantly larger density filed than the average $\mathbb{E}_{\mathcal{T}}[U]$. These samples contribute particularly strong to the high value of TS^{obs}, and hence to the deviation of the test data in relation to the reference model. We may therefore characterize samples with a Z-score higher than some threshold, e.g., samples $x_i \in \mathcal{T}$ with $Z(x_i) > 3$, as the samples that contribute most to the deviation of the test PDF from the reference PDF (see also Fig. 7.1). To get a better understanding in which regions of the feature space the reference model is not compatible with the test data, we may further use a clustering algorithm to identify regions in the feature space with a high density of samples x_i with $Z(x_i) > 3$ [142]. Investigating the high-discrepancy regions can lead to a better understanding of the causes of the deviation from the reference model.

7.5 Application for Model Testing

The statistical setting described in the previous sections is very general and can be used to check the compatibility of any two data sets with each other. Hence, the described testing strategy has a wide range of scientific and engineering applications. Here, we are interested in its application for model testing and the discovery of discrepant regions.

Both, the parametric method using neural networks and the non-parametric method based on k nearest neighbor estimations, have been applied to some simulated toy models [141, 142]. In particular, the non-parametric approach has been applied to detect a (simulated) black matter signal embedded in reference data generated according to the SM of particle physics [142]. The data set $\mathcal{R}$ contains the background events according to the SM, and the data set $\mathcal{T}$ contains also background events but with added black matter signals. It is then demonstrated in [142] that the test statistic based on nearest neighbor estimations is enough sensitive to detect the black matter signals. Furthermore, the same scenario together with added noise to the background simulation is considered in [142]. As expected, such noise lowers the sensitivity to the black matter signals and hence, dependent on the magnitude of the noise, the test may not be sensitive enough to reject the null hypothesis (which is in this case that the SM fully describes the test set). Nonetheless, the method to identify discrepant regions may still be applied and point towards regions in the feature space where the test data deviates the most from the reference data. These are likely the regions where the black matter signal occurs.

We conclude that the work in [141, 142] provides a proof-of-concept for model-independent tests. It demonstrates that the tests have a good sensitivity to various hypothetical new physics signals. Even if the test and the reference model do not deviate in large regions of the feature space (which is common for applications in particle physics), deviations in a small region of the feature space may still be detected by the model-independent tests. We should however keep in mind that real-world data is unlikely as clean as the simulated data and that the suggested methods may require some further improvements to be successfully applied to real data. Furthermore, parametrized test with neural networks tend to overfit for high-dimensional feature spaces with sparse data points, and regularization methods might be needed to avoid this problem [141].

Part III
Representation Learning for Physical Discoveries

In this part, we focus on an essential sub-process of model creation, namely extracting compact representations describing a physical system from experimental data using minimal prior knowledge about physics or maths. To achieve this, we map a simplified version of the physicist's reasoning process, which we refer to as *SciNet*, to an artificial neural network structure and train the network to learn "natural" representations for the given data.

In Chap. 8, we provide a mathematical formulation of the simplified physical reasoning process and introduce several mathematical criteria that a "natural" representation of physical data should satisfy (Sect. 8.4). These criteria are not tailored to a specific physical system, but apply to any physical system in principle. Furthermore, we discuss how building some mathematical or physical prior knowledge into *SciNet*'s structure can help to interpret the found representation in a mathematical framework (Sect. 8.5) or as a classification of physical properties (Sect. 8.6).

In Chap. 9, we describe the artificial neural network structures that allow to find representations of data with the desired mathematical properties. The structures used for this purpose resemble the simplified model we are using to describe a physicist's reasoning process, and we refer to all of these network structures as *SciNet* for simplicity.

We then describe applications of *SciNet* to different physical toy examples from classical as well as quantum mechanics in Chap. 10. The results show that *SciNet* finds the same parameters in its representation that are used by physics textbooks in the considered examples (Sects. 10.2 and 10.6). Furthermore, it is demonstrated that conceptual information can be extracted from the found representations. In Sect. 10.4, it is demonstrated for a collision experiment of two bodies how conservation laws are related to the parameters found in *SciNet*'s representation. In Sect. 10.5.1, it is shown how to use *SciNet* to decide if a set of measurements is sufficient to fully specify the state of a quantum system, and finally, in Sect. 10.7, *SciNet* points towards a heliocentric solar system by switching from angular data of planet orbits collected on Earth to a representation of heliocentric angles. In addition, Sect. 10.3 shows that *SciNet* can linearize the dynamics of the nonlinear pendulum using mathematical prior knowledge about Koopman operator theory. Finally, in Sect. 10.8, *SciNet* is applied to a physical system consisting of several objects to classify the interaction type between particles.

We conclude with some open questions and directions for future work on representation learning for physics in Chap. 11.

Chapter 8
Theory: Formalizing the Process of Human Model Building

In this chapter, the process of physical model creation is formalised. Physical models rely on representations of physical systems. Crucially, we would like to find “natural” representations, since this simplifies the process of building physical models based on the parameters of the representation. We start with introducing a very broad formalization of what we consider to be an experimental setting and observations. Then, we define *minimal representations* (with respect to a set of questions), which incorporate the natural requirement that a representation should be as compressed as possible but still contain the information about the considered physical system that is relevant to answer all questions that might be asked. Further, we introduce two criteria for separating the parameters in a minimal representation in a natural way, leading to the definition of *statistically independent* and *operationally meaningful* representations. The criteria for operationally meaningful representations derive from an operational requirement on minimizing the communication between different physicists to achieve their tasks. In addition, in physics we often prefer representations that evolve in time according to a simple update rule. For example, a linear time evolution of the position x of a particle moving with constant velocity v is described by a simple update rule $x \mapsto x + v\Delta t$ for a time difference Δt (where e.g. storing x^2 in each time step would require a more complex update rule). We formalize this requirement on a representation and also consider a special form of update rules motivated by Koopman operator theory in more detail. Finally, we describe how to find representations of systems consisting of several interacting objects.

8.1 Motivation

As discussed in the introduction, we group the research process of a physicist into five processes: (1) Choosing a physical subsystem; (2) Creating experimental setups and (3) performing measurements; (4) Creating models and (5) testing them. In this

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_8

work, we focus on one part of the creation of models from experimental data, namely, finding “natural” representations of physical systems. This step is crucial to make progress towards an interpretable artificial intelligence agent, because compact and “natural” representations of data build the basis to find the mathematical equations that describe the behavior of the system, and hence to make progress in understanding the laws of our physical environment.

8.2 Physicist’s Reasoning Process

Modeling the complete reasoning process of a physicist is an extremely complex task and way beyond the scope of this book. However, one may split the process of generating models based on experimental data into three parts:

1. Finding the relevant physical parameters necessary to predict the behavior of a system,
2. finding the equations describing the system’s behavior (using the relevant parameters) and
3. generalizing the equations and unify them with already known equations.

Here we focus on finding the relevant physical parameters. As discussed in Sect. 1, we again consider the simplified model *SciNet* for the physical reasoning process given in Fig. 1.1. Recall that an encoder $E : \mathcal{O} \to \mathcal{R}$ maps observations to representations $\mathcal{R}$, followed by a decoder $D : \mathcal{R} \times \mathcal{Q} \to \mathcal{A}$ that maps the representation together with a question to an answer. Crucially, we have left to formalize what we exactly understand to be a “natural” representation. We discuss this in the following sections.

8.3 Experimental Setting and Data Creation

Before we formalize what we understand under a “natural” representation of a physical system, let us explain what we consider to be an “experimental setting” and how it can be used to generate experimental data (following the definition given in [23]). We take a very general view, and represent a state of Nature by a set of hidden parameters[1] $\mathcal{H} \subset \mathbb{R}^n$ for some $n \in \mathbb{N}$. Note that any kind of information can be encoded into a set of real numbers. Then, we define:

- An *experiment* is a mapping S from a space of input parameters $\mathcal{X}$ (think of the setting described by the positions of the dials and buttons in an experiment) to a subset of the hidden parameters $\mathcal{H}' \subset \mathcal{H}$. An *experimental setting* is an instance of an experiment with specified parameters, i.e., the map S together with an $x \in \mathcal{X}$.

[1] The expression “hidden parameters” is not to be confused with the “hidden variables” appearing in Bell’s theorem [159].

- An observation is a stochastic mapping $G : \mathcal{H} \mapsto \tilde{\mathcal{O}}$, where $\tilde{\mathcal{O}} \subset \mathbb{R}^m$ is the set of possible measurement outputs.

We assume that the value of the hidden parameters that do not lie in $\mathcal{H}'$ are constant for different experimental settings. Hence, for different experimental settings, we can consider an observation to be an stochastic mapping (which we again denote by G) from $\mathcal{H}'$ to $\tilde{\mathcal{O}}$. Further, since the hidden parameters $\mathcal{H}$ are not directly observable, it is more operational to consider the concatenated stochastic map $V = G \circ S : \mathcal{X} \mapsto \tilde{\mathcal{O}}$. Nonetheless, for illustrative reasons, we will sometimes use the concept of the hidden parameters in the following sections.

Example Let us again consider a particle moving with constant speed along the x-axis. In an experiment, a physicist has access to three dials that regulate the mass m, the the kinetic energy E and the initial position x_0 of the particle. Crucially, the physicist does not have to know which parameters are regulated by the dials. The hidden parameters would contain the information about all properties of the particle, like the mass, color, shape, and so on. An observer then measures the position $x(t_i)$ of the particle at N different times t_i. Hence, the map V for the considered example would be $V(m, E, x_0) = ((t_0, x(t_0)), \ldots, (t_N, x(t_N)))$ with $x(t_i) = x_0 + vt_i$ where $v = \sqrt{2E/m}$. One could then generate different data samples by choosing different experimental settings, i.e., different settings of the dials regulating m, E and x_0, and observe time series of the position of the particle for each setting.

Since we allow for a stochastic map V, this model also works for theories with probabilistic observations such as quantum mechanics. There, the hidden parameters contain the information of the wave function of a physical system, and the predictions are sampled according to the distribution predicted by the Born rule (see Sect. 10.5 for a detailed discussion). In such cases, one may want to perform a measurement several times for a fixed experimental setting x, since one observation may not provide enough information about the underlying distribution $P_V(\tilde{o} \in \tilde{\mathcal{O}}|x)$ that corresponds to the stochastic map V. Hence, an observation corresponding to an experiment with parameters x may be described by a parametrization of the distribution $P_V(\tilde{o} \in \tilde{\mathcal{O}}|x)$. For example, the distribution P_V of a stochastic map V with two possible outcomes $\tilde{O} = \{0, 1\}$ and $P_V(0|x) = p(x)$, $P_V(1|x) = 1 - p(x)$ for any function $p : \mathcal{X} \mapsto [0, 1]$ can be described by the single parameter $p(x)$. This parameter can be estimated by choosing the experimental setting x and performing the measurement n times. For large n, we then have $p(x) \approx n_0/n$, where n_0 is the number of times the output 0 occurred. We refer to the value set of such parametrizations by $\mathcal{O}$ (and denote it simply by the "set of observations"). In the case of a deterministic mapping V, the distribution $P_V(\tilde{o} \in \tilde{\mathcal{O}}|x)$ ist fully described by its output o with $P_V(o|x) = 1$. Hence, in this case we can set $\mathcal{O} = \tilde{\mathcal{O}}$.

8.4 Criteria for Operationally Meaningful Representations

Have you ever wondered why physicists work with variables like the mass of a particle or the frequency of a pendulum? Sure, we have mathematical formulas using these parameters to predict the systems behavior. But assuming no background in maths, are there more fundamental reasons why we work with these quantities?

In this section we formalize different desirable criteria for "natural" representations of physical systems, based on the ideas introduced in [22, 23]. Later, in Sect. 10, we will demonstrate with several toy examples that these criteria lead to representations commonly used in physical textbooks. Hence, we gain some more fundamental understanding why physicists work with certain variables.

8.4.1 Minimal Representation

One requirement for a representation of experimental data is that it contains sufficient information for answering all questions that are asked about the underlying physical system. Naturally, one would like to have just the necessary amount of information in the representation and remove redundandy. Apart from saving memory, removing redundancy often also simplifies the interpretation of the representation. In the picture described in Sect. 8.3, we would like to find the minimal amount of hidden parameters that are required to correctly reply to the asked questions. In this section, we formalize these criteria for the case of real valued data and parameters that are stored in the representation.

We consider real-valued data $(\mathcal{O}, \mathcal{Q}, a^\star)$, with a set of observations $\mathcal{O} \subseteq \mathbb{R}^r$, a set of questions $\mathcal{Q} \subseteq \mathbb{R}^s$ and as set of answers $\mathcal{A} \subseteq \mathbb{R}^t$, for some $r, s, t \in \mathbb{N}$. Note that the questions are encoded into a sequence of real parameters before being fed to the neural network. Thereby, the actual representation of a single question is irrelevant as long as it enables the network to distinguish questions that require different answers. We recall that the function $a^\star : \mathcal{O} \times \mathcal{Q} \to \mathcal{A}$ maps an observation $o \in \mathcal{O}$ and a question $q \in \mathcal{Q}$ to the correct answer $a \in \mathcal{A}$. Here, we assume that the observations contain enough information to correctly reply to all questions $q \in \mathcal{Q}$. If this would not be the case, the function $a^\star$ would not exist, and hence there could also not be a representation of the observations that contains sufficient information.[2]

[2] If the observations would not contain enough information to reply to all questions, the prediction accuracy of *SciNet* would be theoretically bounded. Hence, even for optimal encoder and decoder mappings E and D, *SciNet* could not always reply correctly to all the questions. Therefore, observing low prediction accuracy after training *SciNet*, one can conclude that the observational data is incomplete (see Sect. 10.5.1 for an example).

Definition 8.1 *Sufficient representation (with smooth decoder) [22]* A representation[3] $\mathcal{R}$ (defined by an encoder mapping $E : \mathcal{O} \to \mathcal{R} \subset \mathbb{R}^p$) for the data described by the triple $(\mathcal{O}, \mathcal{Q}, a^\star)$ is called a *sufficient representation* if there exists a smooth map $D : \mathcal{R} \times \mathcal{Q} \mapsto \mathcal{A}$, such that $D(E(o), q) = a^\star(o, q)$ for all observations $o \in \mathcal{O}$ and questions $q \in \mathcal{Q}$.

This definition asserts that the encoder map E encodes all information of the observation $o \in \mathcal{O}$ that is necessary to reply to all questions $q \in \mathcal{Q}$.

Definition 8.2 *Minimal representation [22]* We call a sufficient representation $\mathcal{R} \subset \mathbb{R}^p$ a *minimal representation* if there is no other sufficient representation $\mathcal{R}' \subset \mathbb{R}^{p'}$ with $p' < p$.

This formalizes what we consider to be a "maximally compressed" representation, i.e., a representation requiring the minimal number of real parameters, but that still encodes all relevant information. Without the assumption that the decoder is smooth, it would, in principle, always be sufficient to have a single latent variable, since a real number can store an infinite amount of information. Hence, methods from standard information theory, like the information bottleneck [38, 160, 161], are not the right tool to give the number of variables a formal meaning. In Appendix A, we use methods from differential geometry to show that the number of variables p in a minimal representation corresponds to the number of degrees of freedom (hence, the hidden parameters in the setting given in Sect. 8.3) in the observation data required to answer all possible questions.

8.4.2 Separation of Physical Parameters

The notion of a *minimal representation* introduced in Sect. 8.4.1 does only consider the number of parameters and the information contained in a representation, however, it remains unclear by which criterion the different parameters should be separated. For example, feeding the decoder time series $(t_i, x(t_i))_{i\in\{1,\ldots,N\}}$ of particles moving with constant velocity, a minimal representation would encode the velocity v and the start position x_0 to be able to predict the future time evolution of the particle's position. A priori, we could store any two parameters a, b from which one can recover the parameters v and x_0. For example, we could store $a = x_0 + v$ and $b = x_0 - v$ in the representation (considering x_0 and v as real numbers with no physical units assigned). It is easy to see that such a representation would still be a minimal representation, since it still contains the full information in two parameters: we recover x_0 as $x_0 = (a + b)/2$ and v as $v = (a - b)/2$. However, for a physicist, it would be much more natural to store x_0 and v as separate parameters. Why do we consider x_0 and v as a "good" choice of variables?

[3] For simplicity, we abuse notation and use the symbol $\mathcal{R}$ to refer to the set of possible values for the representation, as well as to the representation of observations $\mathcal{O}$ that is defined by an encoder mapping $E : \mathcal{O} \to \mathcal{R}$.

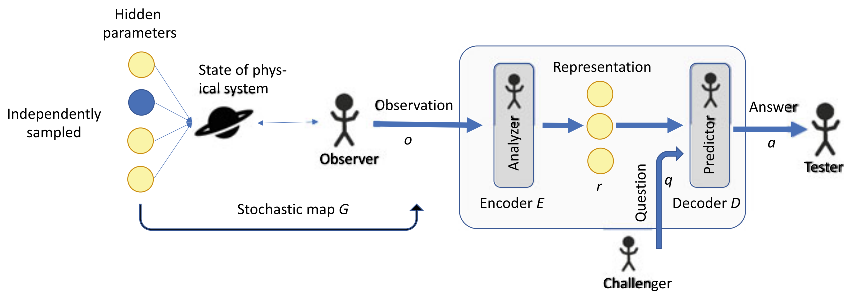


Fig. 8.1 Setting for disentangling statistically independent parameters. We assume that there is a small set of hidden parameters which are sampled independently and fully determine the state of a physical system. Note that the number of parameters (depicted as filled circles) in the figure is not representative. The map G from the hidden parameters to the measurement outputs of an observing agent is allowed to be stochastic. The observing agent then gives the observation o further to an analyzing agent (described by an encoding map E), whose task is to recover the relevant hidden parameters r that are necessary to reply to the questions asked by a challenging agent. The predicting agent that answers a question q based on the representation r is described by a decoder mapping D. Finally, a testing agent compares the answer a of the predicting agent with the correct answer obtained from performing measurements on the environment. Note that in practice, e.g. the analyzer, predictor and challenger could be represented by a single physicist

In the following sections, we introduce different requirements for a "natural" separation of the parameters in the representation. In the context of machine learning, one usually refers to the task of separating variables as *disentangling* latent variables, and to the requirements as *priors*. Later, in Sect. 10 , we will then apply these criteria to different examples and demonstrate that they lead to the separations commonly used by physicists.

8.4.2.1 Statistical Independence

A natural requirement on the parameters in the representation is statistical independence, which is also a common prior in representation learning.

> Different explanatory factors of the data tend to change independently of each other in the input distribution, (...) [27].

Although this criterium usually discovers quite "natural" features in various data structures (see for example [22, 50]), it has the disadvantage that it depends on the sampling method of the training data. Indeed, statistical independence over the training data ensures that knowing one parameter in the representation does not tell us anything about the others. But it remains unclear if this is only true under the specific method that was used to sample the training data or if the found parameters are more fundamentally "independent" features of Nature.

Here we notice that the process of an experimenter collecting data and the process of analyzing the data are mutually dependent on each other. The setting considered in this section is shown in Fig. 8.1, where one assumes that the hidden parameters are sampled independently when creating the observation data. We will consider an operational setting in conjunction with an alternative criterium for disentangling in Sect. 8.4.2.2. This alternative disentangles features based on an operational meaningful criterium that is independent of the sampling strategy used to generate the training data.

Let us now formalize the criterium of independent parameters. We consider real-valued data, which we think of being sampled from an unknown probability distribution. In other words, we assign random variables to the observations O, the questions Q, the latent representation R, and the answers A. We use the convention that a random variable $X = (X_1, \dots, X_{|X|})$ takes samples in $\mathcal{X} \subset \mathbb{R}^{|X|}$, where $|X|$ denotes the dimension of the ambient space of X.

Definition 8.3 *(Statistically independent representations [22])* A representation R (defined by an encoder mapping $E : \mathcal{O} \to \mathcal{R}$) for the data described by the triple $(O, Q, a^\star)$ is called a *statistically independent representation* if the random variables $\{R_1, R_2, \dots, R_{|R|}\}$ are mutually independent.[4]

In praxis, we will never find perfectly independent parameters in the representation. Hence, to quantify the independence of the parameters in the representation, we would like to use a correlation measure $C(R_1, R_2, \dots, R_{|R|})$ with the following properties:

1. $C(R_1, R_2, \dots, R_{|R|}) = 0$ if and only if $R_1, R_2, \dots, R_{|R|}$ are statistically independent,
2. $C(R_1, R_2, \dots, R_{|R|}) \geq 0$ for arbitrary distributions $p_{R_1, R_2, \dots, R_{|R|}}$, and
3. $C(R_1, R_2, \dots, R_{|R|})$ is continuous in the distribution $p_{R_1, R_2, \dots, R_{|R|}}$.

A measure satisfying all of these properties is the total correlation. To define it, we first define the Kullback-Leibler divergence, which can be seen as a distance measure between the distributions of random variables (however, it is not a metric since it does not satisfy the triangular inequality).

Definition 8.4 *(Kullback-Leibler divergence)* For the distributions P_X and P_Y of two continuous random variables X and Y, where P_X is absolute continuous with respect to P_Y, the Kullback-Leibler divergence $D_{\mathrm{KL}}\left(P_X \| P_Y\right)$ is defined as

$$D_{\mathrm{KL}}\left(P_X \| P_Y\right) = \int_{-\infty}^{\infty} P_X(x) \log\left(\frac{P_X(x)}{P_Y(x)}\right) \mathrm{d}x \,. \tag{8.1}$$

Note that we use the convention that $0 \log 0 = 0$, which ensures continuity of the Kullback-Leiler divergence because $\lim_{x \to 0+} x \log x = 0$.

[4] The distribution for the random variables R_i is induced by the distribution over the observations and the encoder mapping E.

Definition 8.5 *(Total Correlation)* For a given set of random variables $\{R_1, R_2, \dots, R_{|R|}\}$, the total correlation $C(R_1, R_2, \dots, R_{|R|})$ is defined as

$$C(R_1, R_2, \dots, R_{|R|}) = D_{\mathrm{KL}}\left(P_{R_1,R_2,\dots,R_{|R|}} \| P_{R_1} P_{R_2} \dots P_{R_{|R|}}\right) . \tag{8.2}$$

Intuitively, the total correlation quantifies the dependency or redundancy among a set of several random variables. It measures the "distance" from the joint distribution of the random variables to the independently distributed random variables with marginal distributions.

The total correlation allows us to summarize the desired and practically applicable properties on a statistically disentangled representation in the following definition.

Definition 8.6 *(Minimally correlated representation)* A representation R for the data described by the triple $(O, Q, a^\star)$ is called a *minimally correlated representation* if and only if:

1. the representation R is minimal and
2. there is no other representation R' with

$$C(R'_1, R'_2, \dots, R'_{|R|}) < C(R_1, R_2, \dots, R_{|R|}) .$$

Remark 8.1 *(Relation to Principal Component Analysis)* One of the most prominent methods to extract uncorrelated features from given data is Principal Component Analysis (PCA). The method performs a linear transformation on the data to extract uncorrelated features (see Chap. 2 and Fig. 2.2). Hence, we could understand PCA as an autoencoder whose encoder (and decoder) perform linear operations. For normal distributed input data, PCA is optimal, in a sense that it recovers the underlying statistically independent features. However, linearly uncorrelated features might still be statistically dependent in general. In such cases, non-linear transformations are required to find the statistically independent features. Since neural networks can in principle approximate any continuous function arbitrarily well (for large enough network sizes) [43, 44], implementing the encoder and decoder with neural networks allows for more complex transformations than PCA.

8.4.2.2 Efficient Communicable

In this section, we present an operational method to disentangle the parameters of a representation that is not biased by the sampling strategy of the data collection process [23]. To illustrate the idea, let us consider the following physical setting: There are two charged particles with masses m_1, m_2 and charges q_1, q_2, respectively. Agent B_1 can perform measurements on the first charge, and agent B_2 on the second charge. The task of agent B_1 is to predict the trajectory of the mass m_1, which depends on m_1, q_1 and q_2, but not on the mass m_2 (and vice versa for agent B_2).

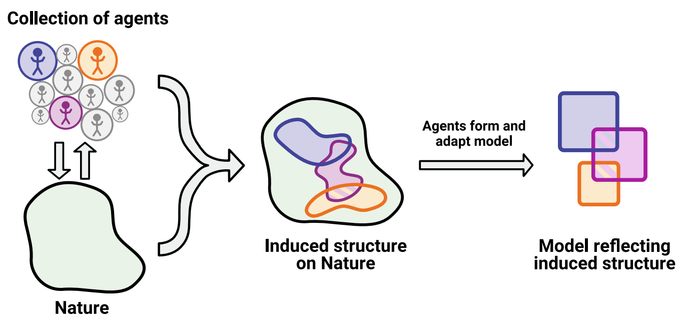

Collection of agents
Nature
Induced structure on Nature
Agents form and adapt model
Model reflecting induced structure

Fig. 8.2 Induced structure on Nature trough communicating agents (Figure taken from Nautrup, Metger and Iten et al., 2020 [23, 237]). Various agents interact with different aspects of Nature through experiments. Different agents may observe or interact with different subsystems, but might not have access to other parts of the environment. To solve certain tasks, agents may require information that can only be accessed by other agents. Therefore, the agents have to communicate. To this end, they encode their information into a representation that can be communicated efficiently to other agents, i.e., they have to find an efficient "language" to share relevant information.This induces structure on the description of Nature by the agents. When the agents build a model of Nature, i.e., learn to parameterize their experimental settings, we want this model to reflect the structure enforced by the requirement that agents compress and communicate parameters in the most efficient way

The agents have to communicate with each other and share information. Instead of sharing all the measurement data, we would like the agents to only share the relevant information. To do so, the agents first create a representation of their data and then share the relevant subset of their parameters that are stored in the representation. This is motivated by human interaction by communicating in an "abstract" language. To minimize the number of parameters the agent B_2 has to communicate to B_1, the charge q_2 should be stored separately from m_2. Indeed, if for example $p_1 = q_2 + m_2$ and $p_2 = q_2 - m_2$ would be stored in two parameters (in some fixed units) instead, the agent B_2 would have to communicate both parameters p_1 and p_2 to agent B_1, such that agent B_1 has enough information to predict the trajectory of the first charge. We conclude that motivating efficient communication in the context of given tasks for different agents poses operationally meaningful structure on the separation of physical parameters, as shown in Fig. 8.2.

We stress that the separation of the parameters depends on the chosen task. If agent B_1 would have to predict $q_2 + m_2$, then it would be optimal for agent B_2 to store the parameters $p_1 = q_2 + m_2$ and $p_2 = q_2 - m_2$. However, from a physical perspective, this task seems not be particularly useful. We define an "operationally meaningful parameter" as one that is useful in various tasks of interest. That is, while the parameters are biased by the tasks we would like to solve, they are not biased by how the measurement data was collected.

Let us formalize this approach. We consider several agents that interact with Nature (by performing experiments and collecting observations) and communicate with each other. The process of how agents could analyze a physical system together

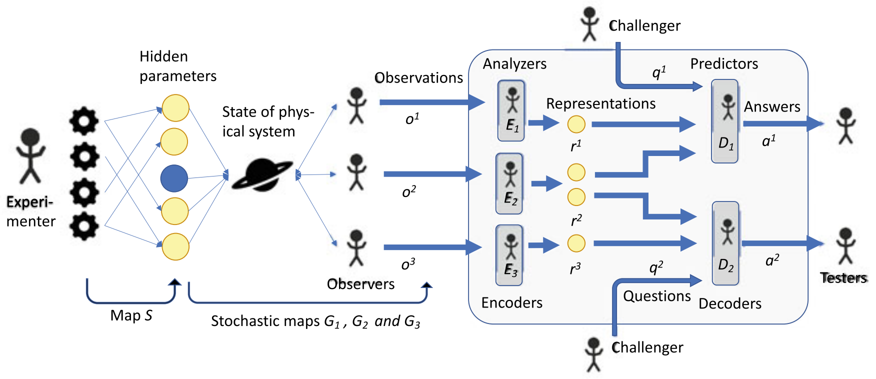


Fig. 8.3 Setting for disentangling operational meaningful parameters. One (or several) experimenters interact with the environment and choose an experimental setting (e.g. by choosing values for dials and buttons). A function S maps such a setting to a set of parameters describing the state of a physical system (the number of parameters depicted is not representative). Three observers have access to different parts of the physical system and based on the stochastic maps G_1, G_2 and G_3, the observations o^1, o^2 and o^3 are sampled. Then, analyzers (given by the encoding maps E_1, E_2, and E_3) produce compressed representations r^1, r^2 and r^3 of observations o^1, o^2 and o^3, respectively. The parameters in the representations are communicated to two predictors (described by the decoding maps D_1 and D_2), whose task is to reply to the questions q^1 and q^2, respectively. In general, some parameters of the representation may be shared with several decoders. The separation of the parameters in the representations is optimized in a way, such that a minimal amount of parameters is required by each of the agents. This criteria for separating parameters is motivated by minimizing the communication between different agents

is shown in Fig. 8.3. A first agent, the experimenter, has chosen an experiment described by the map $S : \mathcal{X} \mapsto \mathcal{H}$.[5] A number of m different observers $G_i : \mathcal{H} \mapsto \mathcal{O}^i$ then collect observations o^i of the physical system for an experimental settings x (and hence, hidden variables $S(x) \in \mathcal{H}$), and we denote the concatenated observations by $o = (o^1, \dots, o^m)$. The set of all concatenated observation is denoted by $\mathcal{O} \subset \mathcal{O}^1 \times \dots \times \mathcal{O}^m$. Note that in general $\mathcal{O}$ is a proper subset of $\mathcal{O}^1 \times \dots \times \mathcal{O}^m$, since the measurement results of different observers may be related to each other. Analyzing agents $E_i : \mathcal{O}^i \mapsto \mathcal{R}^i$ then encode the given experimental data o^i to compressed representations r^i. The analyzing agents then decide which parameters to communicate with whom of the predicting agents. We formalize the selection process as follows: For m analyzing agents, we concatenate all the representations r^i as $r = (r^1, \dots, r^m) \in \mathcal{R} \subset \mathcal{R}^1 \times \mathcal{R}^2 \times \dots \mathcal{R}^m$. Then, for each predicting agent D_j, we define a filter function $\phi_j : \mathcal{R} \mapsto \tilde{\mathcal{R}}^j \subset \mathbb{R}^{d_j}$ that selects a subset of d_j components from the vector $r \in \mathcal{R}$. The predicting agents $D_j : \tilde{\mathcal{R}}^j \times \mathcal{Q}^j \mapsto \mathcal{A}^j$ then get challenged with questions q^j chosen by challengers from sets $\mathcal{Q}^j$ and output answers $a^j \in \mathcal{A}^j$. Testing agents then perform experiments to provide the correct replies

[5] One could also consider several experimenters, however, this setting could be reduced again to the one having only one experimenter without changing the analysis in this work.

$\left(a^j\right)^\star(S(x), q^j)$ for hidden variables $S(x)$ and a question q^j. Intuitively, each filter should only chooses the part of the representation found by the analyzing agents that is required to reply to the questions asked by the challenging agents.

In the example considered at the start of this section, the agent B_1 and B_2 take the role of the observing, analyzing and predicting agents. Concretely, we have two observing agents G_1 and G_2, corresponding to the agents B_1 and B_2, that encode their observations (as analyzing agents E_1 and E_2) into the masses and charges m_1, q_1 and m_2, q_2, respectively. The analyzing agents then communicate m_1, q_1, q_2 and m_2, q_2, q_1 to the predicting agents D_1 and D_2, which again correspond to the agents B_1 and B_2, respectively.[6] In other words, defining the concatenated representation $r = (r_1, r_2, r_3, r_4) = (m_1, q_1, m_2, q_2)$, the two filters ϕ_1 and ϕ_2 map r to (r_1, r_2, r_4) and (r_2, r_3, r_4), respectively. The predicting agents, again corresponding to the agents B_1 and B_2, are then asked by challenging agents (which are different agents than B_1 and B_2) to predict the trajectory of the particles.

Remark 8.2 *(Attention mechanisms)* The formalization of the communication between agents given above could also be interpreted as an *attention mechanism*. Indeed, considering the extreme case where all the agents appearing in the communication scenario are the same, i.e., represent one agent only, the filters could be considered as an attention mechanism that focuses on the information required to answer a specific question. Such mechanisms have been investigated in depth for natural language processing [162–164]. In [162], an attention mechanism is introduced for machine translation. An encoder produces a sequence of annotations representing the information about an arbitrary input sentence. Then, an attention mechanism is used to filter (or weight) the annotations in accordance with their current relevance. The resulting context vector can be used by a decoder to produce a translation. This attention mechanism has been extended to multiple encoder-decoder models for multilingual neural machine translation [163] and summarization [164]. While the methods in these works have a strong focus on improving the outputs of the decoders, in this book, we are mainly interested in how such methods can bring further structure into the latent representation.

To define a *operationally meaningful representation*, we define the concatenation of n questions by $q = (q^1, \ldots, q^n) \in \mathcal{Q} \subset \mathcal{Q}^1 \times \cdots \times \mathcal{Q}^n$, and the concatenation of n answer by $a = (a^1, \ldots, a^n) \in \mathcal{A} \subset \mathcal{A}^1 \times \cdots \times \mathcal{A}^n$. Further, we note that under the assumption that there exists a sufficient representation (i.e., that the data collected by the observing agents is sufficient to reply to all questions), we can consider the function $a^\star : \mathcal{O} \times \mathcal{Q} \mapsto \mathcal{A}$ providing the (concatenated) correct answers a for the (concatenated) questions q as a function of the (concatenated) observations instead

[6] The parameters m_1, q_2 and m_2, q_2 are sent from agent B_1 and B_2, respectively, to themselves. Therefore, on the high-level picture considered at the start of this section, we do not only minimize the communication between B_1 and B_2, but also the "communication" between the agents themselves, i.e., we motivate to minimize the number of parameter that are used to solve the agent's tasks. In this specific case, where the analyzing agents are the same as the predicting agents, minimizing "communication" thus also corresponds to the motivation of the agents to concentrate on the relevant part of their representations (see Remark 8.2).

of the hidden variables. We denote the list $(\phi_1, \dots, \phi_n)$ of filter functions $\phi_j : \mathcal{R} \mapsto \tilde{\mathcal{R}}^j \subset \mathbb{R}^{d_j}$ by ϕ, and the output dimension of ϕ_j by $\dim(\phi_j)$. Before we define an operationally meaningful representation, we need to extend the definition of a minimal representation (Definition 8.2) to multiple encoders. The generalization is straightforward, but it is required because using multiple encoders may increase the number of parameters in a minimal representation compared to the case where one uses one encoder (see Sect. 10.6.2 for an example).

Definition 8.7 *(Minimal representation with multiple encoders)* A representation $\mathcal{R} \subset \mathbb{R}^p$ (defined by encoders $E_1, \dots, E_m$) for the data described by $(\mathcal{O}, \mathcal{Q}, a^\star)$ with $\mathcal{O} \subset \mathcal{O}^1 \times \cdots \times \mathcal{O}^m$ is called a *minimal representation (with m encoders)* if and only if

1. it is sufficient, i.e., there exists a smooth decoder D with $D(r = (r^1, \dots, r^m), q) = a^\star(o = (o^1, \dots, o^m), q)$ for all possible observations $o \in \mathcal{O}$, questions $q \in \mathcal{Q}$, and where we defined $r^i = E_i(o^i) \in \mathcal{R}^i$, and
2. there is no other sufficient representation $\mathcal{R}' \subset \mathbb{R}^{p'}$ with $p' < p$.

Definition 8.8 *(Operationally meaningful representation [23])* A representation $\mathcal{R}$ (defined by encoders $E_1, \dots, E_m$) for the data and filters described by the quadruple $(\mathcal{O}, \mathcal{Q}, a^\star, \phi)$ is called a *operationally meaningful representation* if and only if:

1. the representation $\mathcal{R}$ is minimal and
2. there is no other minimal representation $\mathcal{R}'$ with a selection function ϕ', such that $\sum_j \dim((\phi')^j) < \sum_j \dim(\phi_j)$.

The first requirement ensures that the complete concatenated representation consists of a minimal number of parameters such that it still contains enough information to reply to all questions $q \in \mathcal{Q}$ that might be asked. The second requirement motivates a separation of the parameters in the representation in a way that minimizes the number of parameters sent to each of the decoders.

Remark 8.3 *(SciNet in the context of reinforcement learning)* In this book, we ask *SciNet* to predict different physical quantities based on given observation data collected from a physical environment. From a machine learning perspective, this task is categorized as representation learning. In general, there are tasks that require several interactions with the environment to be solved. For example, the task of moving a ball into a hole by hitting it with a stick (which may be a familiar task for readers paying golf), may require several steps. Further, if we do not hit the hole with the first stroke, it might be unclear how to quantify how "useful" the stroke was. In other words, we may not have a cost function that can be evaluated after each stroke, but we only get a reward if we finally hit the hole. This is a typical setting for reinforcement learning (see Chap. 2). Considering a reinforcement leaning environment and several agents with different tasks, we can combine *SciNet* with techniques from reinforcement learning to construct minimal representation of the state of the environment that is disentangled according to the agent's tasks: Each decoder of *SciNet* can be considered as a separate reinforcement learning agent that has to solve a certain task in its environment with help of additional information received from the representation found by encoders. Instead of learning the representation and the strategy in parallel, it might however be technically simpler to first learn the optimal strategy, and then train *SciNet* with a data set generated by letting a trained agent interact with the environment (see [23] for an example and further details). In other words, the trained agent provides the correct predictions for *SciNet*, and hence a cost function that can be evaluated for each input to *SciNet*.

8.4.2.3 Statistical Independence Versus Operationally Meaningful

In the previous sections, we have described two different methods for disentangling parameters in representation learning. Depending on the situation one approach may be preferable over the other. In operationally meaningful representations, the disentangling is operationally motivated by minimizing the number of parameters communicated between agents. However, to fully disentangle a representation consisting of k parameters, we would need at least k different "natural" questions about the physical system. If we are not aware of sufficiently many such questions, we may instead search for statistically independent parameters, which are biased by the data collection process. Alternatively, one may combine the methods by first disentangling according to questions, and then, disentangle further the resulting sub-representations by motivating statistical independence.

8.4.3 Simple Update Rules

Besides minimality and being disentangled, it is often desirable for physical representations that there are simple rules to obtain predictions from them. For example, it is often desirable that given a representation, there is a simple mathematical formula that evolves the representation in time. To formalize this, let us think about a set $\mathcal{U}$ of (simple) update rules $u : \mathcal{R} \mapsto \mathcal{R}$ acting on a representation $\mathcal{R}$. Further, we consider a specific form of questions $q = (t, \tilde{q})$, where $t \in \mathbb{N}_0$ denotes the number of times the update rule should be applied and the question $\tilde{q} \in \tilde{\mathcal{Q}}$ can be any question referring to the system after applying the updating rule t times.

Definition 8.9 *(Representation with simple update rule [22])* A representation $\mathcal{R}$ for the data described by the triple $(\mathcal{O}, \mathcal{Q} = \mathbb{N}_0 \times \tilde{\mathcal{Q}}, a^\star)$ (generated by an encoder $E : \mathcal{O} \mapsto \mathcal{R}$) is called a *simple representation with respect to a set of update rules* $\mathcal{U}$ if and only if:

1. the representation R is minimal and
2. there is an update rule $u \in \mathcal{U}$ and a decoder mapping $D : \mathcal{R} \mapsto \mathcal{A}$, such that

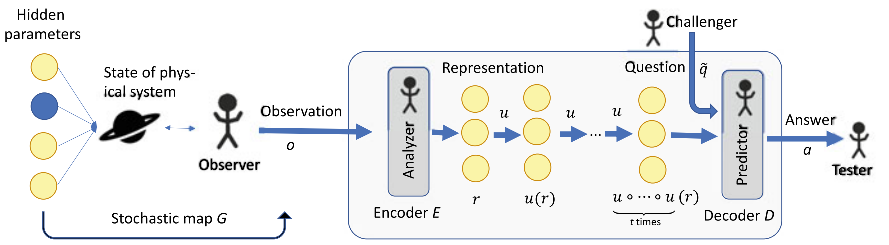


Fig. 8.4 Setting for searching parameters that evolve according to a simple update rule. A small set of hidden parameters fully determine the state of a physical system (the number of parameters depicted in the figure is not representative). The state of the system depends on a parameter $t \in \mathbb{N}_0$ (e.g., the time could correspond to $(\epsilon t + t_0)$s for some $\epsilon > 0$ and initial time t_0). We want to find a representation of the physical system that allows for a simple evolution $u : \mathcal{R} \mapsto \mathcal{R}$ mapping the representation of the system for a parameter t to the representation corresponding to a parameter $t + 1$. Hence, we consider questions of the form $q = (t, \tilde{q})$, i.e., we ask a question $\tilde{q}$ about the state of the system corresponding to the parameter t. To do so, an analyzer encodes an observation o into a representation r, that is then evolved t times by the simple update function u. Then, a decoding agent is asked the question $\tilde{q}$ and his answer a is checked by a testing agent

$$D(\underbrace{u \circ \cdots \circ u}_{t \text{ times}} \circ E(o), \tilde{q}) = a^{\star}(o, q)$$

for all observations $o \in \mathcal{O}$ and questions $q = (t, \tilde{q}) \in \mathcal{Q}$.

8.5 Criteria for Mathematically Meaningful Representations

In Sect. 8.4, we focused on “natural” requirements on representation of physical systems that do not use any mathematical or physical prior knowledge. However, in certain scenarios, we may want to use some mathematical prior knowledge to search for representations that can be interpreted within a mathematical framework. In this section we describe how to use deep learning together with Koopman operator theory [165, 166] to find mathematical descriptions of the dynamical behavior of nonlinear dynamics [167–172]. This investigation serves as an example to demonstrate how mathematical knowledge can be used to describe mathematically meaningful representations.[7]

[7] An alternative method for discovering a representation of a system together with the equations describing its nonlinear dynamics is presented in [173]. The models used in [173] have the fewest terms necessary to describe the dynamics, balancing model complexity with descriptive ability, and thus promoting interpretability and generalizability. Furthermore, they also fit in the framework considered here, i.e., they can be considered as a method to find mathematically meaningful representations.

8.5.1 Koopman Operator Theory

Koopman operator theory was introduced in 1931 in [165, 166]. It is a leading candidate for a systematic approach to linearize nonlinear dynamics [174, 175]. There have been several recent advances in this field, in theory [174–178] as well as in related numerical methods such as dynamic mode decomposition [179–181]. Furthermore, an increasing amount of available measurement data supports applications of the data driven Koopman theory. In the following, we provide a short introduction into Koopman operator theory. For a more detailed recent review on this topic we refer to [182].

We consider a dynamical system whose state is described by a variable $o(t) \in \mathcal{O} \subset \mathbb{R}^n$ at time t and its dynamics by the vector field $f : \mathcal{O} \mapsto \mathbb{R}^n$ together with the differential equation

$$\dot{o}(t) = f(o(t)) \,. \tag{8.3}$$

The discrete time evolution for time Δt can be described by a function $F : \mathcal{O} \mapsto \mathcal{O}$, given by

$$o_{k+1} = F(o_k) \,, \tag{8.4}$$

where $o_k = o(k\Delta t)$ is the state of the system at time $k\Delta t$ with $k \in \mathbb{N}_0$. We are interested in finding the function F that describes the discrete-time dynamics of the considered system from given experimental data. A lot of physical systems of interest allow for a compact representation as differential equations in $o(t)$. However, in most cases apart from linear dynamics, the differential equations are analytically not trackable.

Koopman operator theory does not consider the evolution of the state $o(t)$ of the system, but rather the evolution of observables derived from $o(t)$ and hence described by real valued maps $g : \mathcal{O} \mapsto \mathbb{R}$ on the state space. Let us consider a box containing some ideal gas as an example. The state $o(t)$ of the system is fully described by all the positions and velocities of the particles in the gas. An example for an observable would be the pressure of the gas, which can be seen as a function of the sate $o(t)$. The *Koopman operator*, $\mathcal{K}$, is then the linear and infinite dimensional operator that advances measurement functions for a discrete time step Δt

$$\mathcal{K}g = g \circ F \implies \mathcal{K}g(o_k) = g(o_{k+1}) \,. \tag{8.5}$$

The linearity of the *Koopman operator* is a direct consequence of the linearity of the composition operation because for observables $g_1, g_2 : \mathcal{O} \mapsto \mathbb{R}$, we have

$$\mathcal{K}(g_1 + g_2) = (g_1 + g_2) \circ F = g_1 \circ F + g_2 \circ F = \mathcal{K}g_1 + \mathcal{K}g_2 \,. \tag{8.6}$$

So, the good news are that we replaced non linear dynamics with a linear operator $\mathcal{K}$; the bad news are that the operator $\mathcal{K}$ acts on a infinite dimensional space. To understand the dynamics described by $\mathcal{K}$, it can help to restrict the operator to finite dimensional subspaces that are invariant under the action of $\mathcal{K}$. Any subspace L (over the complex numbers) that is spanned by eigenfunctions of $\mathcal{K}$ is an invariant subspace, because for an n-dimensional subspace spanned by eigenfunctions $\phi_1, \dots, \phi_l : \mathcal{O} \mapsto \mathbb{C}$ with eigenvalue $\lambda_1, \dots, \lambda_l \in \mathbb{C}$ we have

$$\mathcal{K}(\alpha_1\phi_1 + \cdots + \alpha_l\phi_l) = \alpha_1\lambda_1\phi_1 + \cdots + \alpha_l\lambda_l\phi_l \in L\,, \tag{8.7}$$

for any complex coefficients $\alpha_1, \dots, \alpha_l \in \mathbb{C}$. Hence, the Koopman operator restricted to a subspace spanned by eigenvectors allows for a linear finite dimensional representation of the dynamics on this subspace. Since we have a comprehensive theory for linear systems, such a restriction has the potential to enable advanced prediction and control of nonlinear systems.

8.5.2 Representation of Koopman Eigenfunctions

Obtaining eigenfunctions of the Koopman operator in practical applications has proven challenging. In this section we describe (based on [183]) how neural networks can be used to find these eigenfunctions. Searching for Koopman eigenfunctions can be considered as searching for transformations $\phi_1, \dots, \phi_l : \mathcal{O} \mapsto \mathbb{C}$ of observations $o \in \mathcal{O}$, such that $r = (\phi_1(o), \dots, \phi_l(o))$ evolves in time Δt as $r \mapsto (\mathcal{K}\phi_1(o), \dots, \mathcal{K}\phi_l(o)) = (\lambda_1\phi_1(o), \dots, \lambda_l\phi_l(o)) =: \mathcal{K}(\lambda)(\phi_1(o), \dots, \phi_l(o))$, for some complex eigenvalues λ_i and where we defined $\mathcal{K}(\lambda) : \mathbb{C}^l \mapsto \mathbb{C}^l$ by $\mathcal{K}(\lambda)(\alpha_1, \dots, \alpha_l) = (\lambda_1\alpha_1, \dots, \lambda_l\alpha_l)$ for complex numbers $\alpha_1, \dots, \alpha_l$. Defining the (complex valued) encoder $E : o \mapsto r = (\phi_1(o), \dots, \phi_l(o))$, we can consider this setting as a special case of the one considered in Sect. 8.4.3, where we described representations that allow for a simple update rule u (see Fig. 8.4). Indeed, if we set $u = \mathcal{K}(\lambda)$ and choose the questions to be of the form $q = (k, \tilde{q})$, where $\tilde{q}$ is the (fixed) question "What is the state o_k at time $k\Delta t$ of the system?", Definition 8.9 (generalized for complex valued encoders) ensures that the encoder of the corresponding representation implements a minimal number of Koopman eigenfunctions $\phi_1, \dots, \phi_l$, such that the resulting representation $r = (\phi_1(o), \dots, \phi_l(o))$ is sufficient to recover the state o.

A wide range of physical systems requires a Koopman operator with continuous spectrum for its accurate description [166]. For example the frequency of a (non linearized) pendulum, whose angular displacement θ satisfies the differential equation $\ddot{\theta} = -\omega_0^2 \sin(\theta)$ for some $\omega_0 \in \mathbb{R}$, continuously decreases as the energy of the pendulum is increased. In such cases, approximations by a finite sum of a few eigenfunctions might not be accurate enough to be useful for a better understanding of the underlying system. One method that helps to handle continuous spectra and that is described in [183] is to allow the eigenvalues to depend on the representation r,

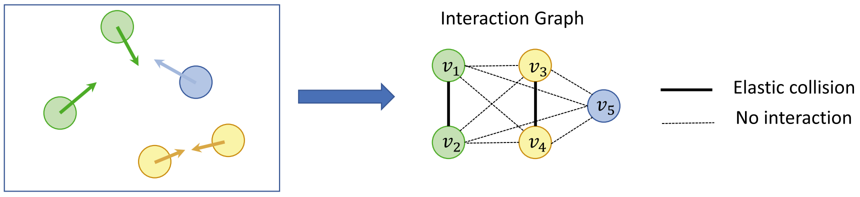


Fig. 8.5 Interaction graph for a toy model of moving balls. Five balls move within a box with different velocities (and collide elastically with the wall of the box). We consider the toy model of balls colliding with the same color elastically, whereas balls of different colors do not interact. From observing the time evolution of the system of the five balls, one can create an interaction graph that represents the information on how the five objects interact with each other. The vertices $v_1, \ldots, v_5$ of the graph represent the balls and the edges between the balls describe their interactions

i.e. we consider a Koopman operator $\mathcal{K}(\lambda)$ with $\lambda \equiv \lambda(r)$. In [183], this method is successfully applied to linearize the dynamics of the nonlinear pendulum (Sect. 10.3) and high dimensional nonlinear fluid flow.

8.6 Criteria for Physically Meaningful Representations

To simplify the interpretation of *SciNet*'s representation and to improve its generalization power, we may sometimes want to build some physical prior knowledge into the machine learning setup. There are a lot of options for the choice of prior knowledge like symmetries or pairwise interactions of subsystems, and also ways of implementation. We focus on an example that should serve as a demonstration of how such prior knowledge can be built into *SciNet* and how it can help to interpret the found representation. In this section we consider systems consisting of several objects (or more generally "parts") that interact with each other. We assume that our observation data is structured in a way that we know how it relates to the objects, in other words, we assume that an observation $o \in \mathcal{O}$ of k objects is of the form $o = (o_1, \ldots, o_k)$ for real valued vectors o_i. Different approaches have been considered to describe the behavior of such systems [184–188]. Here, we focus on the approach described in [184], which does require minimal prior knowledge about how objects interact with each other and on the form of the observation data o_i. Furthermore, it allows to find a description of the interaction structure given as an *interaction graph*. In the following sections, we describe what an interaction graph is and then how to extract a representation of it for dynamical systems.

8.6.1 Interaction Graph

Mathematically, a graph is defined as a tuple $(\mathcal{V}, \mathcal{E})$ of a set of vertices $v \in \mathcal{V}$ and edges $e = (v, v') \in \mathcal{E}$ that can be thought as connections between two vertices $v \in \mathcal{V}$ and $v' \in \mathcal{V}$ (see Fig. 8.5 for an example). Such a structure can naturally describe a set of interacting objects by assigning vertices $v_1, \dots, v_n \in \mathcal{V}$ to the objects and edges $e_{i,j} = (v_i, v_j)$ to the action of the object assigned to v_i on the one assigned to v_j. However, to also describe the properties of the objects and kind of interactions, we work with embeddings of real vectors h_i and $h_{i,j}$ of the vertices and edges, respectively. For example, let us consider a toy model with n balls with positions x_i, velocities w_i and colors c_i. Hence, the embeddings of the vertices could simply be $h_i = (x_i, w_i, c_i)$. Let us assume that there are only two types of interactions between the balls: two balls with the same color collide fully elastically, whereas balls with different colors do not interact at all. Hence, the embedding of the edges could be $h_{i,j} = h_{j,i} = 1$, if $c_i = c_j$ and $h_{i,j} = h_{j,i} = 0$ if $c_i \neq c_j$. The interaction graph for such a system is shown in Fig. 8.5.

8.6.2 Representation of Interaction Graph

The goal of this section is to formalize the process of extracting the type of interactions, i.e., the embeddings $h_{i,j}$ of the edges of the interaction graph, from observation data of a dynamic system (following the approach in [184]). The number of objects may thereby vary between different observations. Let us consider time sequences $o_i = [x_i(t_l)]_{l \leq T} = x_i(t_1), \dots, x_i(t_T)$ of length T of the d-dimensional states $x_i(t_l) \in \mathbb{R}^d$ of the object with label i at fixed times $t_1, \dots, t_T$.

We would like to infer the structure of the interaction graph corresponding to the interactions between the objects described by the observations $[x_i(t_l)]_{l \leq T}$. This process can be considered to have the basic structure of *SciNet* as shown in Fig. 1.1. However, usually we consider real valued observations of fixed dimension, whereas here the set of observations $\mathcal{O}$ is the set of sequences of time sequences $[x_i(t_l)]_{l \leq T} \in \mathbb{R}^d \times \mathbb{R}^T$, hence $\mathcal{O} \subset \{[s_i]_{i \leq k} : s_i \in \mathbb{R}^d \times \mathbb{R}^T, k \in \mathbb{N}\}$. Now, an encoder should be able to map observation data of any number of objects to a representation that describes the interactions of the objects, and hence whose size scales quadratically in the number of objects k. More formally, the encoder should map an observation $o \in \mathcal{O}$ for k objects to a representation r that encodes all embeddings $h_{i,j}$ with $i, j \in \{1, \dots, k\}$ and $i \neq j$. Assuming that the interactions can be fully characterized by an e-dimensional real vector $h_{i,j} \in \mathbb{R}^e$, we then have $r \in \mathbb{R}^{k(k-1)} \times \mathbb{R}^e$. For simplicity, we use double indices for the representation such that $r_{i,j} = h_{i,j}$. Note that such an encoder with variable input and output size can not trivially be implemented using a feedforward neural network. Instead, one can use *graph neural networks* (GNNs) to handle these difficulties, as described in Sect. 9.7.

We then ask *SciNet* to predict the future time evolution of all the objects, given an initial state $o(t_0) = o_1(t_0), \ldots, o_k(t_0)$ at some time t_0 and based on the found representation encoding the kind of interactions between all the objects. To be able to predict the time evolution with high accuracy far into the future (such that all the objects will interact once), *SciNet* must have inferred all the necessary information about the interactions and stored this information in the latent representation r. Hence, such a structure can be used to learn conceptual information about which objects interact with each other and which parameters describe these interactions. We describe in Sect. 9.7 how such representations can be found with GNNs [184].

Chapter 9
Methods: Using Neural Networks to Find Simple Representations

In the previous chapter, we have formalized what we consider to be a "simple" representation of physical data by introducing the notion of minimal *statistically independent* and *operpationally meaningful* representations, as well as representations allowing for a simple update rule and representation of systems consisting of several interacting objects. In this chapter, we discuss how neural networks can be used to find these kind of representations directly from a given data set (following the ideas described in [22, 23]). Apart from the given data and the theoretical requirements on representations found in the previous chapter, we focus on minimizing any further prior knowledge built into the machine learning system. Hence, the network architectures introduced in this chapter can be applied to a wide range of data.

9.1 Motivation

We would like to build software that is able to learn the different kinds of representations described in Sect. 8.4 directly from given data. To achieve this, the main difficulty is to find a function that encodes the observations into a representation that satisfies the properties formalized in Sects. 8.4, 8.5 and 8.6. One crucial property we require for any representation is that it is *sufficient*, i.e., that it contains enough information to reply to a set of questions. To check if a representation is sufficient, in general one has to search over decoding functions that map the representation and a question to an answer (Fig. 1.1). If one finds a decoder that outputs the correct answers for all questions, one can be sure that the found representation contains all necessary information. We conclude that we have to search for an encoder and decoder in parallel. This can be efficiently done by using artificial neural networks, which are a powerful tool to learn functions directly from data. Note that for many applications of neural networks, it is crucial that they can generalize to input samples that significantly differ from the samples seen during the training. However, for this work, neural networks are mainly used as a function-approximating tool.

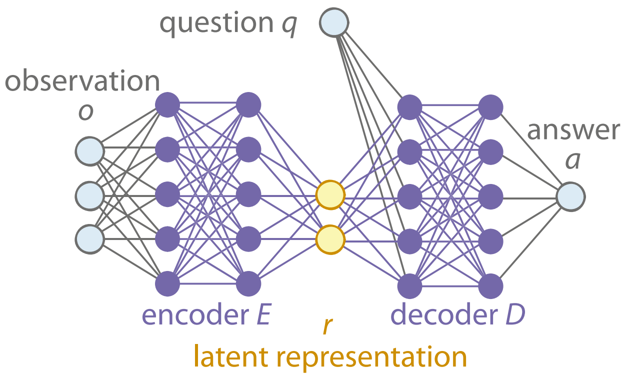


Fig. 9.1 Network structure for *SciNet* motivated by the human physical reasoning process (Figure taken from Iten and Metger et al., *Physical Review Letters*, 2020 [22]). Observations are encoded as real parameters fed to an encoder (a *feed-forward neural network*), which compresses the data into a representation (*latent representation*). The question is also encoded in a number of real parameters, which, together with the representation, are fed to the decoder network to produce an answer (The number of neurons depicted is not representative.)

9.2 General Network Structure to Learn Representations

The modelling process described in Sect. 8.2 can be translated directly into a neural network architecture (Fig. 9.1), which we refer to as *SciNet* in the following [22]. The encoder and decoder are both implemented as feed-forward neural networks. The resulting architecture, except for the question input, resembles an autoencoder in representation learning [27, 189], and more specifically the architecture in [190]. Note that if there is a finite set of questions, one could in principle replace the question input by asking the decoder to reply to all the questions in each run. However, when working with continuous variables describing the questions, such a strategy would not work.

During the training, we provide triples of the form $(o, q, a^\star(o, q))$ to the network, where $a^\star(o, q) \in \mathcal{A}$ is the correct reply to question $q \in \mathcal{Q}$ given the observation $o \in \mathcal{O}$. With this we train the encoder E_Θ and the decoder D_Φ, where Θ and Φ denote the collection of weights and biases of the encoder and the decoder, respectively. The training process of *SciNet* to learn minimal representations is summarized in Box 1. To find (an estimate of) the minimal number of parameters in the representation, one simply raises the number of parameters until the answer quality of *SciNet* does no longer increase significantly.[1] In this way, a representation consisting of a minimal

[1] One may want to use more efficient search algorithms over the number of parameters, in particular in the case where an initial estimate on the minimal number is given.

number of parameters can be found. As discussed later, there are more efficient methods to achieve this (see Remark 9.2). The learned parameterization (defined through the encoder E_Θ) is typically called latent representation [27, 189]. It is crucial that the encoder is completely free to choose a latent representation itself, instead of us imposing a specific one. Because neural networks with at least one hidden layer composed of sufficiently many neurons can approximate any continuous function arbitrarily well [191], the fact that the functions E and D are implemented as neural networks does not significantly restrict their generality. In other words, we consider significantly large networks E_Θ and D_Φ, so that there is always a choice of parameters Θ and Φ, such that E_Θ and D_Φ are close to the optimal functions for the encoder and the decoder. However, unlike in an autoencoder, the latent representation need not describe the observations completely; instead, it only needs to contain the information necessary to answer the questions posed.

Box 1: Train *SciNet* to learn minimal representations (Section 9.2)

Goal: Find a minimal representation for a given data set $\mathcal{D}$ consisting of triples of the form $(o, q, a^\star(o, q))$, where $a^\star(o, q) \in \mathcal{A}$ is the correct reply to question $q \in \mathcal{Q}$ given the observation $o \in \mathcal{O}$.

Network Structure: Shown in Figure 9.1. We denote the parametrized encoder mapping by $E_\Theta : \mathcal{O} \mapsto \mathcal{R}$, and the decoder mapping by $D_\Phi : \mathcal{R} \times \mathcal{Q} \mapsto \mathcal{A}$.

Cost Function: For a triple $(o, q, a^\star(o, q))$, the cost is given by $\mathrm{d}\left[D_\Phi(E_\Theta(o), q), a^\star(o, q)\right]$, for some (smooth) distance measure $\mathrm{d}[\cdot, \cdot]$.

Splitting of the data set: We split the data set $\mathcal{D}$ into a training, validation and a test set.

Training process: To find a minimal representation, we start with zero latent neurons and successively increase the number of latent neurons. For each number of latent neurons, we train the network with the training data and validate the precision of the predictions with the validation data. When the accuracy no longer increases significantly, the increase in the number of latent neurons is stopped. [A more efficient method to find the minimal number of required parameters is presented in Box 2.]

Testing: We test the performance of the trained *SciNet* using the test set.

Remark 9.1 In some cases, the neural network E_Θ implementing the encoder may have difficulty finding the perfect encoding. Either because it is limited in expressing the optimal encoding function (because the number of hidden neurons is not large enough or the encoding function is not continuous as demonstrated in Sect. 10.5.1) or because the training process does not converge to a solution that is close to the optimal encoding. In such cases, an alternative way to estimate the minimum number of parameters is to apply the Levina-Bickel algorithm [192].

Extracting information from the network. The architecture in Fig. 9.1 allows us to extract knowledge from the neural network: the relevant information is stored in the representation, and the size of this representation is small compared to the total number of degrees of freedom of the network. This helps to interpret the learnt representation. Specifically, we can compare *SciNet's* latent representation to a hypothesized parameterization to obtain a simple map from one to the other. If we do not even have any hypotheses about the system at hand, we may still gain some insights solely from the number of required parameters or from studying the change in the representation when manually changing the input, and the change in output when manually changing the representation (as in e.g. [49]).

9.3 Network Structures for Separating Parameters

A lot of work in representation learning focuses on methods to disentangle representations, i.e., to store "natural" features in separate neurons in the latent space (see e.g. [49, 193]). Usually, these works focus on the construction of a representation of the full observation input. In contrast, *SciNet* focuses on a representation that is sufficient to reply to all the questions that might be asked. Nonetheless, it is straightforward to use the methods introduced in the machine learning literature to separate parameters in *SciNet*'s representation. However, most of these methods depend on the distribution over the training data, and new methods were developed in [23] for operational disentangling, i.e., methods to disentangle parameters relevant for solving different physical tasks (see Sect. 8.4.2.2 for the details).

9.3.1 Statistically Independent

As described in Sect. 8.4.2.1, one may define a representation to be disentangled if the parameters in the representation are statistically independent over the given data set. To motivate *SciNet* to find minimal statistically independent representations, we could in principle use the same technique as described in Box 1 and add the total correlation $C(R_1, \dots, R_{|R|})$ between the latent neurons as an additive term to the cost function. However, this training technique would be computationally extremely expensive. There are two reasons for this: Firstly, retraining *SciNet* for different number of latent neurons is costly. Secondly, and more importantly, estimating the total correlation during the training is computationally extremely expensive, since this measure depends on the distribution over the latent neurons, which in turn requires a lot of samples to be well approximated.

Box 2: Train *SciNet* to learn statistically independent representations (Section 9.3.1)

Goal: Find a statistically independent representation for a given data set $\mathcal{D}$ consisting of triples of the form $(o, q, a^\star(o, q))$, where $a^\star(o, q) \in \mathcal{A}$ is the correct reply to question $q \in \mathcal{Q}$ given the observation $o \in \mathcal{O}$.

Network Structure: Essentially the one shown in Figure 9.1 (see Figure B.1 in Appendix B for details). We denote the parametrized encoder mapping by $E_\Theta : \mathcal{O} \mapsto \mathcal{R}$, and the decoder mapping by $D_\Phi : \mathcal{R} \times \mathcal{Q} \mapsto \mathcal{A}$.

Desired Cost Function (computational infeasible): For $x = (o, q, a^\star(o, q))$, the desired cost function would be $\mathcal{L}(x) = \mathrm{d}\left[D_\Phi(E_\Theta(o), q), a^\star(o, q)\right] + \alpha\, C(R_1, \ldots, R_{|R|})$, for some (smooth) distance measure $\mathrm{d}[\cdot,\cdot]$, a hyperparameter α, and where $R_1, \ldots, R_{|R|}$ are the random variables for the latent neurons (determined by the encoder E_Θ and the distribution over the observations used as training data).

Computational Feasible Cost Function: β-VAE cost function given in (B.1) in Appendix B.

Splitting of the data set: We split the given data set $\mathcal{D}$ into a training, validation and test set.

Training process: To find a statistically independent representation, we choose a number of latent neurons that is expected to be larger or equal than the number requires for a minimal representation. We train *SciNet* with the β-VAE cost function, where we have to optimize the hyperparameter β. If we still have a high prediction error for the validation data, we may have chosen too few latent neurons in the beginning, and we repeat the process with a higher number of latent neurons.

Testing: We evaluate the prediction loss of the trained *SciNet* using the test set. Further, we would like to evaluate the total correlation of the latent neurons over the test set. This is computational feasible, because we only have to calculate it once after training. If the total correlation is still high, we could train *SciNet* again with a higher value of β to motivate the disentangling of the latent neurons more strongly.

Because of these difficulties, we follow the approach taken in [22] here: We use the techniques of beta-VAEs [49], i.e. of variational autoencoders (see Appendix B) with an adapted cost function that is meant to help disentangle the latent representation. In practice, beta-VAEs often find representations with statistically independent latent neurons. Further, if there are unnecessarily many latent neurons, superfluous neurons tend to be set to a constant zero-activation. However, since there is no guarantee that beta-VAEs always find statistically independent parameters, we verify this a posteriori by calculating the total correlation of the latent variables after finishing the training of *SciNet*. Since the details of how beta-VAEs work will not be relevant to understand this thesis, we explain them in Appendix B. The training process is summarized in Box 2.

9.3.2 Operationally Meaningful

In this section, we describe the approach introduced in [23] to find operationally meaningful representations (see Definition 8.8). Recalling the process given in

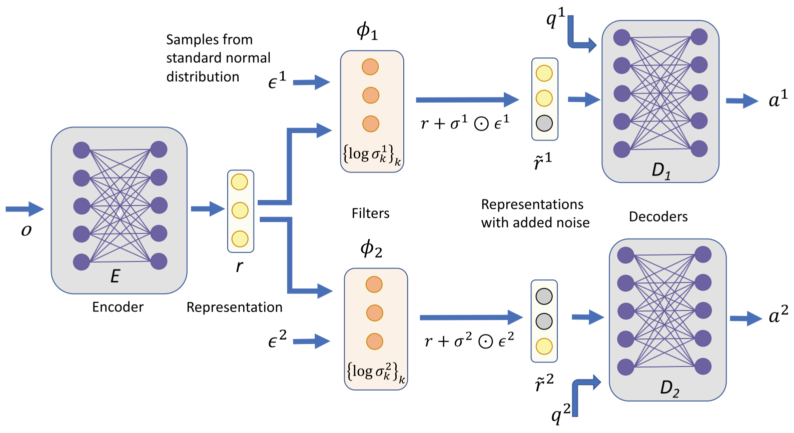


Fig. 9.2 Network architecture to learn operationally meaningful representations. An encoder gets an observation o and outputs a representation r. (In principle, one could consider several encoders, but for simplicity we only use one here). Then, for each decoder D_j, a filter ϕ_j is applied to the representation that adds Gaussian noise to the latent neurons r_k with a standard derivation σ_k^j. The sampling is done using the *reparameterization trick* [193], i.e., in each run of the network we sample $\epsilon_k^j \sim \mathcal{N}(0, 1)$ and provide these samples as an input to the filters. Then, the values $\tilde{r}_k^j = r_k + \sigma_k^j \epsilon_k^j$ are given to the decoder D_j, whose task is to output the response to the question q^j. The number of parameters and decoders in the figure is not representative. Technically, the filters are described by the trainable parameters $\{\log \sigma_k^j\}_k$. If a filter ϕ_j adds a lot of noise to a latent neuron k, there is essentially no information about the value r_k transmitted to the decoder D_j by $\tilde{r}_k^j$. Hence, such neurons (marked in grey in the figure) are considered to be filtered out by ϕ_j, i.e., the information contained in r_k^j is not accessible by the decoder D_j

Fig. 8.3, it is straightforward to replace the encoders and decoders with neural networks. The challenging part of the implementation are the filter functions. Indeed, these functions have to select which values of the latent neurons are sent to which of the decoders. Hence, they are of a binary nature, where for each latent neuron the filter can be "on" or "off" for a certain decoder D_j, meaning that the value of this latent neuron is or is not sent to the decoder D_j. To train such filters, we would like to "smooth" them, so that we can propagate gradients trough them while running the back-propagation training algorithm. We can smooth the filters based on the reparametrization trick [193], as described in the following. For simplicity, let us assume that we work with a single encoder. The implementation described below is summarized in Fig. 9.2.

Box 3: Train *SciNet* to learn operationally meaningful representations with a single encoder (Section 9.3.2)

Goal: Find operationally meaningful representations for a given data set $\mathcal{D}$ consisting of triples of the form $(o, (q^1, q^2, \dots, q^n), (a_1^\star(o, q^1), a_2^\star(o, q^2), \dots, a_n^\star(o, q^n))$, where $a_j^\star(o, q^j) \in \mathcal{A}^j$ is the correct response to question $q^j \in \mathcal{Q}$ given the observation $o \in \mathcal{O}$.

Network Structure: Shown in Figure 9.2. We denote the parametrized encoder mapping by $E_\Theta : \mathcal{O} \mapsto \mathcal{R}$, the filters by $\phi_{\sigma^j} : [0, 1]^n \times \mathcal{R} \mapsto \mathcal{R}$, where n denotes the number of latent neurons $\dim(\mathcal{R})$, and the decoder mappings by $D_{\Omega^j} : \tilde{\mathcal{R}}^j \times \mathcal{Q}^j \mapsto \mathcal{A}^j$.

Cost Function For a triple $x = (o, (q^1, \dots, q^n), (a_1^\star(o, q^1), \dots, a_n^\star(o, q^n))$, noise samples $\epsilon^j = (\epsilon_1^j, \dots \epsilon_n^j)$ with $\epsilon_k^j \sim \mathcal{N}(0, 1)$, and outputs $a^j = (a_1^j, \dots, a_{\dim(\mathcal{A}^j)}^j)$ for a decoder D_{Ω^j}, i.e., $a^j = D_{\Omega^j}\left[\phi_{\sigma^j}\left(\epsilon^j, E_\Theta(o)\right), q^j\right]$, the cost is given by

$$\mathcal{L}(x, \{\epsilon^j\}_j) = \sum_j \mathrm{d}\left[a^j, a_j^\star(o, q^j)\right] - \beta \sum_k \log(\sigma_k^j)\,, \tag{9.1}$$

for some (smooth) distance measure $\mathrm{d}[\cdot, \cdot]$ and a hyper parameter β that regulates the tradeoff between good accuracy for the predictions and minimizing the number of latent neurons required by each decoder.

Splitting of the data set: We split the given data set $\mathcal{D}$ into a training, validation and test set.

Training process: To find a minimal representation, we start with zero latent neurons and successively increase the number of latent neurons. For each number of latent neurons, we train the network with the training data and the cost function given above, where we have to optimize the hyperparameter β, and validate the precision of the predictions with the validation data. If we achieve nearly perfect precision, we stop the process of increasing the number of latent neurons. (A more efficient method is described in Remark 9.2).

Testing: We evaluate the prediction loss of the trained *SciNet* using the test set.

Implementation of filters [23]. To train a filter in a smooth way, we need to be able to make a smooth transition from the "on" to the "off" state of a filter. Clearly, if the filter is "on" for a latent neuron, all its information is transferred to the decoder, where in the "off" state, no information about the parameter stored in the latent neuron is given to the decoder. If we add noise to the latent neuron, we can smoothly regulate how much information about the latent parameter is given to the decoder. Indeed, let us formally consider the latent neuron k to be sampled from a Gaussian distribution $\mathcal{N}(r_k, \sigma_k^j)$, where r_k is the value of the k-th latent neuron (output from the encoder) and σ_k^j determines the amount of noise added to the latent parameter that is subsequently given to the decoder D_j. For the decoder D_j, the value of σ_k^j has the following meaning:

1. A large value of σ_k^j (relative to the variation of the k-th latent neuron over the observation data) means that we add a lot of noise to the value r_k of the k-th latent neuron. Hence, the decoder D_j gets essentially no information about r_k and we consider this case to be a filter that does not select the k-th latent neuron.
2. A value of σ_k^j that is close to zero means that essentially all information about r_k is transferred to the decoder D_j, so the filter selects this latent neuron.

Although we can smoothly regulate the noise, we have to sample from the Gaussian distribution in each run of the network. Sampling is a non-differentiable process in the parameters in the set Θ, describing the weights and biases of the encoder. We can use the *reparameterization trick* introduced in [193] (for variational auto encoders) to solve this problem: we replace the sampling operations by using auxiliary random numbers $\epsilon_k^j \sim \mathcal{N}(0, 1)$ as inputs to the neural network. Then, the k-th latent variable with the filter applied for a decoder D_j and distributed with $\mathcal{N}(r_k, \sigma_k^j)$ can be generated by $r_k + \sigma_k^j \epsilon_k^j$. Using vector notation, we get the complete filter output for decoder D_j as a vector $r + \sigma^j \odot \epsilon^j$, where $r = (r_1, \ldots, r_m)$, $\sigma^j = (\sigma_1^j, \ldots, \sigma_m^j)$ and $\epsilon^j = (\epsilon_1^j, \ldots, \epsilon_m^j)$ for m latent variables, and where we use the symbol "$\odot$" for component-wise multiplication. Sampling ϵ_k^j does not interfere with the gradient descent because ϵ_k^j is independent of the trainable parameters Θ.

Note that in our implementation (Fig. 9.2), we use $\log(\sigma_k^j)$ as trainable parameters of the filters instead of σ_k^j, because $\log(\sigma_k^j)$ can take on positive or negative real values, whereas σ_k^j must always be positive. Hence, this replacement simplifies the training, because we do not have to ensure positivity of these values in each training step. In addition, the values of σ_k^j can get very large during training and the logarithm maps these values into a reasonable range.

Cost function for minimizing the information transmitted by filters. As stated in Item 2 of Definition 8.8, we would like to minimize the number of "on"-states for the filters to minimize the information communicated between agents (see Sect. 8.4 for a detailed explanation of the motivation). With the implementation above, this translates into minimizing $-\sum_{j,k} \log(\sigma_k^j)$. Indeed, this cost term motivates the network to increase the noise added to the latent neurons and competes with the cost term motivating the decoders to output correct predictions. The full cost function and a description of the training process is given in Box 3.

Remark 9.2 (Minimizing the total number of latent neurons) If for a certain k, the value of σ_k^j is large for all j, the information of the k-th latent neuron is not transmitted to any of the decoders and hence could be ignored. Since we would like to find a minimal representation, we would like to maximize the number of latent neurons that are not used by any decoder under the condition that all the decoders are still able to output accurate predictions. The cost function term $-\sum_{k,j} \log(\sigma_k^j)$ does not necessarily motivate *SciNet* to minimize the total number of latent neurons. Indeed, assume that the representation has two latent neuron r_l and r_m that store the same information, and that two decoders D_i and D_j require this information. Then, the cost is the same wether the value of r_l (or r_m) is transferred to both decoders or r_m is transferred to D_i and r_l to D_j (or vice versa). To minimize the number of latent neurons, we could train *SciNet* with an increasing number of latent neurons (as described in Box 3), but this method is computationally expensive. Alternatively we could add a *global filter* parametrized by σ_k^{global} that selects a number of latent neurons before they are sent to all of the decoder specific filters ϕ_j. The cost term $-\sum_{k,j} \log(\sigma_k^j) - \gamma \sum_k \log(\sigma_k^{\text{global}})$ for some hyperparameter γ then motivates to minimize the total number of latent neurons (by motivating the global filter to select a minimal number of neurons that are sent further to the decoders). Although this method is quite efficient, it turns out to be less reliable than increasing the number of latent neurons one by one in numerical experiments, in the sense that the network gets harder to train and gets stuck more often in sub-optimal local minima. Nonetheless, the method is useful to get an estimate on the required number of latent neurons. Starting from this estimate, one can successively lower the number of latent neurons until the prediction accuracy of *SciNet* significantly decreases. When this happens, the current number of latent neurons plus one is likely the minimal number required.

9.4 Network Structure to Find Representations with Simple Update Rules

In this section, we describe how to find representations that evolve according to a simple update rule (see Definition 8.9), based on the ideas described in [22]. Recall that a decoder D is asked to reply to a question $\tilde{q} \in \tilde{\mathcal{Q}}$ after evolving the representation found by an encoder E for $t \in \mathbb{N}_0$ steps according to a simple update rule u chosen from a set $\mathcal{U}$ (Fig. 8.4). Again, it is straightforward to implement the encoder and the decoder appearing in this setting with neural networks. To speed up the training process, we ask the network to reply to a question $\tilde{q}$ after each application of the update rule. Hence, we ask the questions $(0, \tilde{q}), \ldots, (t, \tilde{q})$ in each training step (Fig. 9.3). This provides more feedback to the network and helps it to learn more quickly.

It remains to investigate how to implement the update rule. Let us assume that the functions $u \in \mathcal{U}$ are smooth. There are two technical cases that we have to distinguish:

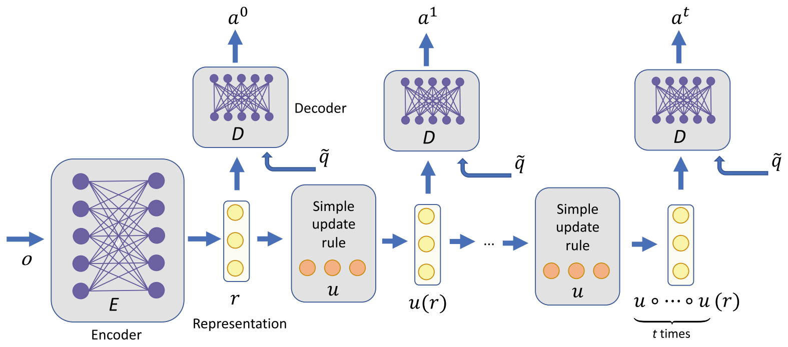


Fig. 9.3 Network architecture to learn representations that evolve according to a simple update rule. An encoder maps an observation o to a representation r that allows for a simple update rule u (think of evolving a representation of a physical system for a fixed time interval). A decoder should correctly respond to a question $\tilde{q}$ in each step. The set of questions is therefore described by the tuples $(0, \tilde{q}), \ldots, (t, \tilde{q})$, where the first element of the tuple determines the step at wich question $\tilde{q}$ is asked. The answers given by the decoder are labeled with the step number as $a^0, \ldots, a^t$. Crucially, the same decoder is used in each step. In the case where u evolves the system in time, and where $\tilde{q}$ asks to predict a physical quantity at a given time, using only one decoder means that the physical laws determining how to predict physical quantities do not change with time. The network architecture shown in the figure is similar to the one of recurrent neural networks [194]

1. The set of updates rules $\mathcal{U}$ can be smoothly parametrized by a set of parameters Γ, i.e., there is an open connected set of parameters $\Gamma \subset \mathbb{R}^l$ for some $l \in \mathbb{N}$, such that $u_\gamma(r) \in \mathcal{U}$ depends smoothly on $\gamma \in \Gamma$ for any $r \in \mathcal{R}$ and such that the sets $\{u_\gamma\}_{\gamma\in\Gamma}$ and $\mathcal{U}$ are in one-to-one correspondence.
2. The set of update rules cannot be smoothly parametrized.

In case 1, we can propagate gradients trough the update rule during the training of the neural network. Hence, the parametrized update rule u_γ can be considered as a part of the neural network, where the parameters γ are the trainable parameters analogous to the weights and biases of artificial neurons.

The training in case 2 is usually more costly and there are a lot of tricks in the machine learning literature to map it back to case 1. If, however, no such trick works for a given case, a brute-force strategy to tackle case 2 for a finite set $\mathcal{U}$ would be the following: For each $u \in \mathcal{U}$, train an encoder and a decoder to optimize *SciNet*'s prediction accuracy. If one finds a u, such that *SciNet* can predict nearly perfectly, one is done. However, if there are a lot of elements in $\mathcal{U}$, such a strategy gets computationally intractable. In the case where the set $\mathcal{U}$ can be partitioned into a small number of smoothly parametrizable subsets, one could handle these subsets separately by the strategy applied in case 1.

Box 4: Train *SciNet* to learn representations with simple update rules that can be smoothly parametrized (Section 9.4)

Goal: Find a representation for a given data set $\mathcal{D}$ consisting of triples of the form $(o, [(0,\tilde{q}), \ldots, (t,\tilde{q})], [a^\star(o,(0,\tilde{q})), \ldots, a^\star(o,(t,\tilde{q}))])$, where $a^\star(o,(s,\tilde{q}))$ is the correct reply to question $\tilde{q} \in \mathcal{Q}$ in step $s \in \mathbb{N}_0$ given the observation $o \in \mathcal{O}$.

Network Structure: Shown in Figure 9.3. We denote the parametrized encoder mapping by $E_\Theta : \mathcal{O} \mapsto \mathcal{R}$, the parametrized update rule by $u_\gamma : \mathcal{R} \mapsto \mathcal{R}$, and the decoder mapping by $D_\Omega : \mathcal{R} \times \tilde{\mathcal{Q}} \mapsto \mathcal{A}$.

Cost Function For a triple $x = (o, [(0,\tilde{q}), \ldots, (t,\tilde{q})], [a^\star(o,(0,\tilde{q})), \ldots, a^\star(o,(t,\tilde{q}))])$ and answers provided by the decoder D_Ω given as

$$a^s = D_\Omega(\underbrace{u_\gamma \circ \cdots \circ u_\gamma}_{s \text{ times}} \circ E_\Theta(o), \tilde{q})\,,$$

the cost is given by

$$\mathcal{L}(x) = \sum_s \mathrm{d}\left[a^s, a^\star(o,(s,\tilde{q}))\right]$$

for some (smooth) distance measure $\mathrm{d}[\cdot,\cdot]$.

Splitting of the data set: We split the given $\mathcal{D}$ into a training and test set.

Training process: To find a minimal representation, we start with zero latent neurons and successively increase the number of latent neurons. For each number of latent neurons, we train the network (including the update rule u_γ) with the training data and the cost function given above. If we achieve nearly perfect precision, we stop the process of increasing the number of latent neurons. Alternatively, one could try a more efficient method to find the minimal number of parameters, such as the ones used for beta-VAE [49] (see Appendix B).

Testing: We evaluate the prediction loss of the trained *SciNet* using the test set.

The resulting network structure shown in Fig. 8.4 is similar to standard *recurrent neural networks* [194] which are used to process sequential data in machine learning and have a wide range of applications such as machine translation [195–197].

9.5 Optimality Guarantees on the Representation

In the previous sections we introduced different methods to find "natural" representations. However, as common for methods based on neural networks, in general we do not have a guarantee that the suggested methods find the optimal representation. Nonetheless, some properties of a representation can be checked *a posteriori*:

- Sufficiency of the representation (by checking the prediction accuracy on the test data[2]).
- Statistical independence of the latent parameters (by calculating the total correlation of the latent parameters).
- The representation allows for a simple update rule (by checking the prediction accuracy after evolving the representation for several steps).

On the other hand, it is more challenging to test the following properties:

- Minimality of the representation.
- Optimality of the filters, i.e., minimality of the number of parameters that are communicated to the decoders in the setting for searching operationally meaningful representations (see Sect. 9.3.2).

These properties are defined trough the absence of any other sufficient representation that requires less parameters. Hence, to check minimality in a straightforward way, one would have to compare the found representation with all possible representations, which are infinitely many. Instead, one may for example reduce the number of latent neurons and train *SciNet* several times. If the prediction accuracy stays low, this points towards the fact that the number of latent neurons we started with is minimal. On the other hand, investigating the correlation between the latent neurons as well as the effect of varying their value on the predictions may help to better understand the found representation.

9.6 Network Structure to Find Koopman Eigenfunctions

As an example for mathematically meaningful representations, we considered representations that allow for linear dynamics in Sect. 8.5. In the case of a discrete spectrum of the Koopman operator, the setting considered there can be seen as a

[2] If the observations are very noisy, it might be more complex to check sufficiency of a representation.

special case of searching for representation with simple update rules. Hence, we can use the same methods as introduced in Sect. 9.4, which correspond to the methods used in [183] in this case. The only difference to Sect. 9.4 is that we are searching for complex valued representations. Since we can only represent real values with (standard) artificial neurons, we represent the complex valued latent variables $\tilde{r}_i \in \mathbb{C}$ by two real variables $r_{2i-1} \in \mathbb{R}$ and $r_{2i} \in \mathbb{R}$ corresponding to the real and imaginary part of $\tilde{r}_i$. Recall from Sect. 8.5 that each such pair corresponds to the output $\phi_i(o)$ of an eigenfunction ϕ_i of the Koopman operator for an observation input o. Further, we may write the eigenvalue for the eigenfunction ϕ_i as $\lambda_i = \exp\left[(\mu_i + \omega_i i)\,\Delta t\right]$ for coefficients $\mu_i, \omega_i \in \mathbb{R}$ and a discrete (fixed) time step $\Delta t > 0$, and denote the stacked coefficients as $\mu = (\mu_1, \ldots, \mu_m)$ and $\omega = (\omega_1, \ldots, \omega_m)$, respectively. Then, the Koopman operator $\mathcal{K}(\lambda) \equiv \mathcal{K}(\mu, \omega)$ acting on the (real) latent representation $r = (r_1, \ldots, r_{2m})$ consists of Jordan blocks $\mathcal{B}(\mu_i, \omega_i)$ of the form

$$\mathcal{B}(\mu_i, \omega_i) = \mathrm{e}^{\mu_i \Delta t} \begin{bmatrix} \cos \omega_i \Delta t & -\sin \omega_i \Delta t \\ \sin \omega_i \Delta t & \cos \omega_i \Delta t \end{bmatrix}. \tag{9.2}$$

Hence, we have found a smooth parametrization of the set of update rules $\{u_\gamma\}_{\gamma\in\Gamma} = \{\mathcal{K}(\mu, \omega)\}_{\mu\in\mathbb{R}^m, \omega\in[0,2\pi)^m}$. Instead of considering μ and ω as coefficients, one may allow them to depend on the state of the representation r before it gets updated. As discussed in Sect. 8.5.2, this dependence is investigated in [183] to describe the dynamics of nonlinear system with a continuous spectrum of the corresponding Koopman operator. The dependence of $\mu(r)$ and $\omega(r)$ on r can be simply integrated into the network structure shown in Fig. 9.3 by adding a neural network that gets the representation r as an input and provides μ and ω as an output. To enforce circular symmetry in the eigenfunction coordinates, one may provide $\|r\|_2^2$ as an input to the additional network instead of r itself [183].

9.7 Network Structure to Find Interaction Graphs

In Sect. 8.6, we described interaction graphs of dynamical systems. Here, we investigate how to find such graphs with neural networks following the approach in [184]. As noted in Sect. 8.6, we have to implement an encoder with a variable input and output size. Such an encoder has to respect the object structure of the considered system, otherwise it could not generalize from a system consisting of k objects to one consisting of $k + 1$ objects. One class of networks that are well adapted to inputs of a graph structure, are *graph neural networks* (GNNs) [111, 198, 199], which will be introduced in the next section.

9.7.1 Graph Neural Networks

Graph neural networks (GNNs) [111, 198, 199] allow to handle graph structures as an input and have found a wide range of applications, e.g. for predicting chemical properties [111], for control and inference for physical systems [184–186, 200–205] and for investigating social networks [206, 207]. Their expressive power was investigated in [208, 209]. In this section, we give a brief introduction to graph neural networks. There is a lot of flexibility in the structural details of graph neural networks, and here we focus on the structure used in [184]. The described structure can be used to infer the interaction structure of dynamical systems consisting of several objects under the assumption that there is a finite set of e interactions, i.e., working with discrete embeddings $h_{i,j}$ of the interactions between the objects with label i and j.

We consider a graph with embeddings h_i of its vertices. The basic idea for graph neural networks is to update these embeddings several times according to rules that depend on the graph structure, i.e., the update rule for the embedding h_i depends on how the corresponding vertex is connected to other vertices within the graph. Such update rules can be considered as message passing: the neighboring vertices of h_i send their information to h_i, which updates its values according to some rule dependent on the received information. Importantly, such a rule should be commutative, in the sense that the updated value does not depend on the order of how messages are proccessed. After L such updating steps, the updated embeddings h_i^L do not only contain information about the vertices, but also about the graph structure. If we would for example want to predict a binary property of the graph (such as ful connectedness), the embeddings h_i^L can be fed to a neural network, which is asked to classify the graph. Importantly, the neural network can only successfully classify if the update rule applied to the embeddings h_i are of a form, such that the resulting embeddings h_i^L will contain the necessary information about the graph. For a more thorough introduction to graph neural networks, we refer to [210].

Here, we are interested in inferring the type of interactions between objects from observation data of their time evolution. Hence, we also consider embeddings $h_{i,j}$ of the edges between vertices with labels i and j. The update rules are then split into two steps: i) $v \mapsto e$: updating the embeddings $h_{i,j}$ of the edges according to the values of the neighboring vertices, and ii) $e \mapsto v$: updating the embeddings h_i of the vertices according to the incoming edges $\mathcal{N}_i = \{h_{j,i}\}_j$. More concretely, we consider the update rules [210]

$$v \mapsto e: \quad h_{i,j}^l = f_e^l([h_i^l, h_j^l, y_{i,j}])\,, \tag{9.3}$$

$$e \mapsto v: \quad h_i^{l+1} = f_v^l([\sum_{j\in\mathcal{N}_i} h_{j,i}^l, y_i]) \;, \tag{9.4}$$

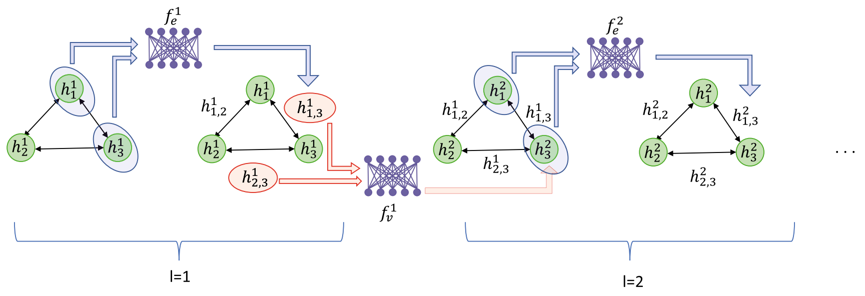


Fig. 9.4 Graph neural network. The figure shows a fully connected graph consisting of three vertices with embeddings h_1^1, h_2^1 and h_3^1. The symbol "$\leftrightarrow$" labels two directed edges pointing in opposite direction. The update rule given in (9.3) is applied on the first graph to get the embeddings $h_{i,j}^1$ of the edges with $i, j \in \{1, 2, 3\}$ and $i \neq j$. Note that we only show the edges $h_{i,j}^l$ with $i < j$ for simplicity. The update rule (9.3) takes the vertices h_1^1 and h_3^1 and auxiliary vertex features (which are not shown in the figure) as an input. Then, it applies a neural network representing the function f_e^1 to the inputs to obtain the edge embedding $h_{1,3}^1$. The other edge embeddings are found analogously (not shown in the figure). Then, the update rule (9.4) is implemented by applying a neural network f_v^1 to the vertex-embeddings $h_{2,3}^1$ and $h_{1,3}^1$ as well as to some auxiliary edge features to obtain the updated vertex embedding h_3^2. Again, the other vertices are updated analogously. This process is then repeated for $L \in \mathbb{N}$ times

where $h_{i,j}^l$ and h_i^{l+1} are the embeddings of the edges and vertices after l update steps and where $y_{i,j}$ and y_i encode initial or auxiliary edge and vertex features as for example node inputs and edge type. A GNN for these update rules is illustrated in Fig. 9.4. In the next section we describe how such GNNs can be used to learn the interaction types between objects from time series data.

9.7.2 Network Structure to Learn Interaction Graphs

In this section, we describe how to implement a network structure to learn the interaction types between objects as described in Sect. 8.6 and based on [184]. We start by constructing an encoder using GNNs that maps an observation $o = [o_1, \ldots, o_k]$ of k objects with $o_i = [x_i(t_l)]_{l \leq T}$ to discrete variables $r_{i,j}$ representing the interaction type. For any labels i and j, the variable $r_{i,j}$ is a one-hot encoding of one of the e distinct interaction types.

Encoder: We use a GNN with the updates rule given in (9.3) and (9.4) for the encoder, where we start from a fully connected graph (see Fig. 9.5). A feedforward neural network f_{emb} is used to map the observations o_i to the vertex embeddings, i.e., $h_i^1 = f_{\text{emb}}(o_i)$. Then, we use the update rules (9.3) and (9.4) (with trivial auxiliary features $y_{i,j}$ and y_i) to evolve the GNN for L steps and obtain $h_{i,j}^L$ as an output.

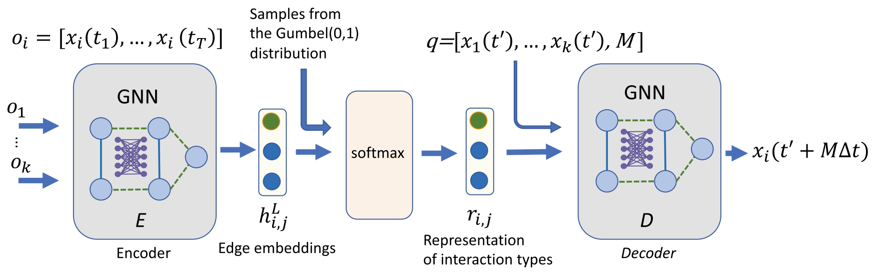

$o_i = [x_i(t_1), \dots, x_i(t_T)]$
Samples from the Gumbel(0,1) distribution
$q=[x_1(t'), \dots, x_k(t'), M]$
o_1
o_k
GNN
E
Encoder
$h^L_{i,j}$
Edge embeddings
softmax
$r_{i,j}$
Representation of interaction types
GNN
D
Decoder
$x_i(t' + M\Delta t)$

Fig. 9.5 Network structure to find interaction types. The encoder is a GNN with update rules given in (9.3) and (9.4). It takes time sequences of k objects as an input, where k may vary between different samples o, and provides edge embeddings $h^L_{i,j}$ as an output. The edge embeddings can be considered to parametrize a distribution, from which we sample continuous approximations of one-hot embedding $r_{i,j}$ of the interaction types. The stochastic input for this reparametrization trick is thereby provided by i.i.d. samples from the Gumbel(0,1) distribution. A decoder is then asked to evolve the state of all the objects starting from initial states $x_i(t')$ and based on the interaction types encoded in the representation r. The decoder is thereby implemented as a GNN with update rules given in (9.7) and (9.8)

Let us denote all parameters describing the GNN, and hence the encoding map, by Θ. To interpret the continuous output variables $h^L_{i,j}$ as discrete variables, we use a common trick in machine learning, namely considering $\text{softmax}(h^L_{i,j})$ as a distribution $q_\Theta(r_{i,j}|o)$ over the discrete variable $r_{i,j}$, where the i-th component of the vector valued function $\text{softmax}(a)$ of an n-dimensional real vector a is defined as

$$\text{softmax}(a)_i = \frac{e^{a_i}}{\sum_{j=1}^n e^{a_i}} . \tag{9.5}$$

Since $\sum_{i=1}^n \text{softmax}(a)_i = 1$, we can indeed consider the output of $\text{softmax}(a)$ as a distribution over a discrete variable with n distinct values.

Sampling: Since we obtain a distribution over the representation as an output of the encoder, we have to sample from it to provide an input to the decoder. Unfortunately, sampling is a non-differentiable process in the parameters of the set Θ, and hence disallows the training of the GNNs with stochastic gradient descent. In Sect. 9.3.2, we encountered the same problem and solved it with the *reparameterization trick* [193]. The essence of this trick is to replace the sampling process by using a differentiable function of the output variables of the encoder and a random variable with fixed distribution. The samples for the random variable with fixed distribution are then provided as an input to the network. However, such a refactoring does not work for discrete random variables due to the discontinuous nature of discrete states. A recent approach to solve this problem was considered in [211], where it is suggested to approximate the discrete distribution with a continuous one and then use the reparameterization trick. More concretely, we draw samples from the distribution

$$r_{i,j} = \text{softmax}\left(\frac{h^L_{i,j} + g}{\tau}\right), \tag{9.6}$$

where $g \in \mathbb{R}^e$ is a vector of i.i.d. samples drawn from a Gumbel(0,1) distribution and the softmax temperature τ controls the strength of the smoothening. The Gambel(0,1) distribution is defined by the probability density function $f(x) = \exp(-x - \exp(-x))$. Note that, for $\tau \to 0$, the distribution converges to one-hot samples of our categorial distribution.

Decoder: The decoder takes the representation $r_{i,j}$ of the interactions as an input and a question $q = [x(t'), M]$ where $x(t') = (x_1(t'), \dots, x_k(t'))$ describes the states of all objects at a time t' and where M is the number of time steps for which the decoder should make the state evolve for a fixed time difference Δt. Let us denote the future states predicted by the decoder for times t', $t' + \Delta t$, ..., $t' + M\Delta t$ by $x_i^0, \dots, x_i^M$ for the i-th object, and let us initialize $x_i^0 = x_i(t')$ for all objects. We use again a GNN for the decoder with a different neural network for each type of interaction. Hence, we slightly adapt the update rules [210]:

$$v \mapsto e: \quad \tilde{h}^c_{i,j} = \sum_{i,j,s} r_{i,j,s} \tilde{f}^s_e([x^c_i, x^c_j]), \tag{9.7}$$

$$e \mapsto v: \quad x^{c+1}_i = x^c_i + \tilde{f}_v(\sum_{i \neq j} \tilde{h}^c_{j,i}), \tag{9.8}$$

where $r_{i,j,s}$ denotes the s-th element of $r_{i,j}$. The first update rule determines the interaction between the two objects with states x^c_i and x^c_j. In the case where the representation $r_{i,j}$ is a one-hot encoding whose a-th entry is equal to one (i.e., $r_{i,j,a} = 1$), only one decoder is applied to the input states and we have $\tilde{h}^c_{i,j} = \tilde{f}^a_e([x^c_i, x^c_j])$. Hence, there is indeed one decoder for each interaction type. The second update rule takes in a description of the interaction $\tilde{h}^c_{j,i}$ between the objects with labels i and j at the c-th time step and should provide the states x^{c+1}_i of the system as an output. Hence, applying these update rules M times should approximate the time evolution of the objects for time $M\Delta t$.

Training: The complete network shown in Fig. 9.5 can be trained with stochastic gradient descent. The loss function consists of two terms. A prediction loss given by

$$\sum_{i=1}^{k} \sum_{c=1}^{T} \left\| x_i(t' + c\Delta t) - x^c_i \right\|^2, \tag{9.9}$$

and a loss term motivated by the KL term appearing in the cost function of variational autoencoders (Appendix B) given by a sum of entropies [184]

$$\sum_{i \neq j} H(q_{\Theta}(r_{ij}|o)) \tag{9.10}$$

Chapter 10
Applications: Physical Toy Examples

In this chapter, we apply the network structures of Chap. 9 (to which we refer as *SciNet* for simplicity) to find representations of (simulated) observation data of six physical toy systems. The results demonstrate that *SciNet* can help to recover concepts in physics by providing the relevant physical parameters, both in quantum- and classical-mechanical settings. We also show that the results are robust against noise in the experimental data.

In summary, without providing any prior knowledge about the considered physical systems to *SciNet* (apart from the observational data), we find: (i) given a time series of the positions of a damped pendulum, *SciNet* can predict future positions with high accuracy and it uses the relevant parameters, namely frequency and damping factor (which are sampled independently to generate the training data), separately in two of the latent neurons (and sets the activation of unnecessary latent neurons to zero); (ii) given a time series of the angular displacement and velocity of a non-linear pendulum, *SciNet* switches to a representation that evolves linearly in time. (iii) *SciNet* finds and exploits conservation laws: it uses the total angular momentum to predict the motion of two colliding particles; (iv) given measurement data from a simple quantum experiment, *SciNet* can be used to determine the dimension of the underlying unknown quantum system and to decide whether a set of measurements is tomographically complete, i.e., whether it provides full information about the quantum state. Furthermore, operationally meaningful representations of mixed quantum states can separate local degrees of freedom in the representation from non-local ones; (v) considering observations coming from different experiments with charged particles, *SciNet* separates the charges and masses of the particles in its operationally meaningful representation; (vi) given a time series of the positions of the Sun and Mars as observed from Earth, *SciNet* switches to a heliocentric representation—that is, it encodes the data into the mean anomalies of the two planets with respect to the Sun; (vii) given time series of the positions and velocities of several particles in a box, where some of them are connected by a spring, *SciNet* classifies the interaction between the particles, i.e., *SciNet*'s representation encodes wether two particles are connected by a spring or not. The runtime to train the networks for all examples takes at most a few hours on a standard laptop.

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_10

10.1 Motivation

In Chap. 8, we have formalized what we consider to be a “simple” representation of a physical system and in Chap. 9, we described how to find such representations with neural networks. In this chapter, we apply these methods to several examples and analyze the extracted representations. The results show that *SciNet* finds, without having been given any prior information about the specific physical systems, the same quantities that we use in physics textbooks to describe the different settings. Hence, our formalization of “simple” representations corresponds well to what representations physicists prefer to describe data.

In all the examples, the training data we use is operational and could be generated from experiments, i.e., the correct answer is the one observed experimentally.[1] Here, we use simulations instead because we only deal with classical and quantum mechanics, theories whose predictions are experimentally well tested in the relevant regimes. One might think that using simulated data would restrict *SciNet* to rediscovering the theory used for data generation. However, in particular for quantum mechanics, we are interested in finding conceptually different formulations of the theory with the same predictions.

The specific examples considered in this chapter were chosen according to the following criteria:

- The total number of parameters that needs to be stored in the representation is small (i.e., around 5 real parameters).
- The examples should demonstrate the “usefulness“ of the different criteria for representations defined in Sect. 8.4.
- Since our longterm goal is to apply the methods from Chap. 9 to better understand quantum mechanics, we also consider examples including quantum systems.
- We focus on examples where the representation contains conceptually interesting information.

The first criterium is purely technical and ensures that the interpretation of the representation does not get too cumbersome and that the training of the neural networks does not take too long on a standard laptop. Further examples, e.g. using video recordings of fire flames as observations, can be found in [192].

From a machine learning perspective, all the considered examples are similarly challenging. The Tensorflow library [212] makes setting up the network structures very simple. Usually, we work with encoders and decoders consisting of two hidden layers with around 200 neurons per layer (see Table C.1 for the details). Networks of this size were able to approximate the encoding and decoding mappings well for all the following examples.[2] The remaining difficulty was to find a “good” choice

[1] For example, one could consider a temporal sequence of positions as non-operational data, but use video recordings instead, since such observations are similar to those that humans perceive with their eyes. Although this is a valid argument, using video data as observations conceptually adds nothing to the examples considered in this chapter, but would complicate neural network training.

[2] A simple method to choose the network size is the following: Start with a small network size for *SciNet* (with enough latent neurons) and train it with the only target of minimizing the prediction

for the hyperparameters such as the parameter β appearing in the cost function 9.1 to learn operationally meaningful representations. Importantly, one does not have to know a minimal representation of the considered physical system to choose the hyperparameters (see Boxes 2 and 3). Although, as discussed in Sect. 9.5, the methods introduced in Chap. 9 are not guaranteed to find the "optimal" representation, they always find them in the following examples (without post-selecting examples, i.e., we did not post-select examples based on whether *SciNet* worked or not).

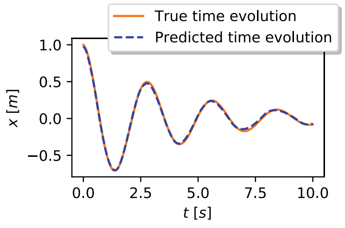
10.2 Damped Pendulum

Following the work in [22], we consider a simple example from classical physics, the damped pendulum, described in Box 5. The time evolution of the system is given by the differential equation $-\kappa x - b\dot{x} = m\ddot{x}$, where κ is the *spring constant*, which determines the frequency of the oscillation, and b is the *damping factor*. We keep the mass m constant (it is a scaling factor that could be absorbed by defining $\kappa' = \kappa/m$ and $b' = b/m$), such that κ and b are the only variable parameters. We consider the case of weak damping here, where the solution to the equation of motion is given in Box 5.

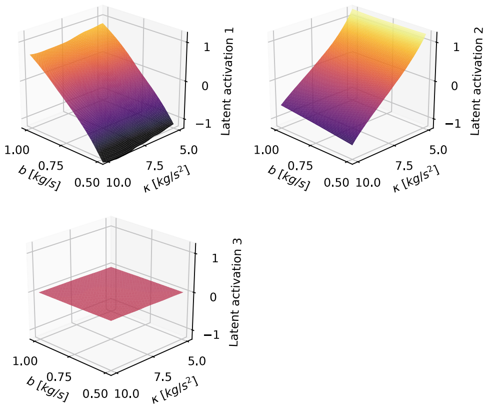
We generate the training data by uniformly sample the spring constant κ and the damping factor b. Hence, we would expect that a statistically independent representation (Definition 8.3) recovers these parameters. We choose a network structure for *SciNet* with 3 latent neurons. As an input, we provide a time series of positions of the pendulum and we ask *SciNet* to predict the position at a future time (see Box 5 for details). The training process is described in Box 1 and based on beta-VAE [49]. The accuracy of the predictions given by *SciNet* after training is illustrated in Fig. 10.1a.

Without being given any physical concepts, *SciNet* learns to extract the two relevant physical parameters from (simulated) time series data for the x-coordinate of the pendulum and to store them in the latent representation. As shown in Fig. 10.1b, the first latent neuron depends nearly linearly on b and is almost independent of κ, and the second latent neuron depends only on κ, again almost linearly. Indeed, more quantitatively, using the methods introduced in [213–215] and implemented in [216] to estimate the total correlation $C(R_1, R_2)$ (which corresponds to the mutual information in this case) between the two latent neurons, we find $C(R_1, R_2) = 0.107$, where in comparison we have $C(R_1, R_1) = 10.95$ for fully correlated latent neurons. We conclude that *SciNet* has recovered the same time-independent parameters b and κ that are used by physicists. The third latent neuron is nearly constant and does not provide any additional information—in other words, *SciNet* recognized that two parameters suffice to encode this situation.

loss. If the prediction accuracy stays low, increase the size of the network. Successively go on like this until good prediction accuracy can be achieved. (If the prediction accuracy stays low even for large networks, the input data might not contain sufficient information).



(a)



(b)

Fig. 10.1 Damped pendulum (Figures taken from Iten and Metger et al., *Physical Review Letters*, 2020 [22]). *SciNet* is fed a time series of the trajectory of a damped pendulum. It learns to store the two relevant physical parameters, frequency and damping, in the representation, and makes correct predictions about the pendulum's future position. **a** Trajectory prediction of *SciNet*. Here, the spring constant is $\kappa = 5$ kg/s^2 and the damping factor is $b = 0.5$ kg/s. *SciNet's* prediction is in excellent agreement with the true time evolution. **b** Representation learned by *SciNet*. The plots show the activations of the three latent neurons of *SciNet* as a function of the spring constant κ and the damping factor b. The first two neurons store the damping factor and spring constant, respectively. The activation of the third neuron is close to zero, suggesting that only two physical variables are required. On an abstract level, learning that one activation can be set to a constant is encouraged by searching for unctorrelated latent variables, i.e., by minimizing the common information of the latent neurons during training

Box 5: Time evolution of a damped pendulum [22] (Sect. 10.2)

Problem: Predict the position of a one-dimensional damped pendulum at different times.
Physical model: Equation of motion: $m\ddot{x} = -\kappa x - b\dot{x}$.

Solution: $x(t) = A_0 e^{-\frac{b}{2m}t}\cos(\omega t + \delta_0)$, with $\omega = \sqrt{\frac{\kappa}{m}}\sqrt{1 - \frac{b^2}{4m\kappa}}$.

Observation: Time series of positions: $o = \left[x(t_i)\right]_{i\in\{1,\ldots,50\}} \in \mathbb{R}^{50}$, with equally spaced $t_i \in [0, 5]s$. Mass $m = 1$kg, amplitude $A_0 = 1$m and phase $\delta_0 = 0$ are fixed; spring constant $\kappa \in [5, 10]$ kg/s^2 and damping factor $b \in [0.5, 1]$ kg/s are independently and uniformly sampled during data creation.

Question: Prediction times: $q = t_{\text{pred}} \in [0, 10]s$.

Correct answer: Position at time t_{pred}: $a^\star = x(t_{\text{pred}}) \in \mathbb{R}$.

Network structure: Essentially, the network depicted in Fig. 9.1 with three latent neurons (see Appendix B for the detailed description of the network structure). The network size can be found in Table C.1.

Training: Described in Box 2 with parameters given in Table C.2.

Key findings:

- *SciNet* predicts the positions $x(t_{\text{pred}})$ with a root mean square error below 2% (with respect to the amplitude $A_0 = 1$m) (Fig. 10.1a).
- *SciNet* stores κ and b in two of the latent neurons, and does not store any information in the third latent neuron (Fig. 10.1b). Hence, it finds a statistically independent representation.

10.3 Dynamics of the Nonlinear Pendulum

In this section we demonstrate how mathematical prior knowledge integrated into *SciNet*'s network structure can help to investigate the dynamical behaviour of a nonlinear pendulum. The angular displacement θ of a nonlinear pendulum satisfies the differential equation $\ddot{\theta} = -\omega_0^2 \sin(\theta)$ with $\omega_0 = \sqrt{g/l}$, where g is the acceleration due to gravity and l denotes the length of the pendulum. This differential equation has complex analytic solutions that were described in terms of Jacobi elliptic functions in 2007 [217]. Koopman operator theory takes a different approach by searching for a coordinate transformation of $(\theta, \dot{\theta})$, such that the transformed coordinates allow for a linear time evolution (see Sect. 8.5). Here, we describe the results from [183], demonstrating how *SciNet* can be used to find such coordinate transformations.

The training data for *SciNet* is generated by sampling θ and $\dot{\theta}$ from the interval $[-0.5, 0.5]$. The network structure for *SciNet* with 2 latent neurons is described in Sect. 9.6. As an input to *SciNet*, we provide the angle and the angular velocity at a starting time and ask *SciNet* to predict the time evolution of these quantities (see Box 6 for the details). The training process is similar as described in Box 4 and we refer to [183] for the details.

Box 6: Time evolution of a nonlinear pendulum [183] (Sect. 10.3)

Problem: Predict the angle and angular velocity of a one-dimensional nonlinear pendulum at different times.

Physical model: Equation of motion: $\ddot{\theta} = -\omega_0^2 \sin(\theta)$ with $\omega_0 = \frac{1}{\text{s}}$.
Solution: An analytic solution can be derived in terms of Jacobi elliptic functions [217].
The data set used for training and evaluation of the neural network is created by solving the equation of motion in MATLAB using the *ode45* solver [183].

Observation: $o = [\theta_0, \dot{\theta}_0]$ with $\theta_0, \dot{\theta}_0 \in [-0.5, 0.5]$.

Question: Implicit.

Correct answer: Time series $\left[a_0, \ldots, a_n\right] = \left[(\theta(t_0), \dot{\theta}(t_0)), \ldots, (\theta(t_n), \dot{\theta}(t_n))\right]$ of $n = 50$ observations, with fixed time steps $\Delta t = t_{i+1} - t_i$ of 0.02 seconds.

Network structure: Essentially, the network structure depicted in Fig. 9.3 with two latent neurons and parametrized update rules of the form

$$u_{\mu,\omega}(r_1, r_2) = \mathrm{e}^{\mu \Delta t}([\cos(\omega \Delta t) r_1 - \sin(\omega \Delta t) r_2], \sin(\omega \Delta t) r_1 + \cos(\omega \Delta t) r_2]),$$

where μ and ω are given as an output of an additional neural network $\Lambda : \mathbb{R} \mapsto \mathbb{R}^2$ that takes $r_1^2 + r_2^2$ as an input (to enforce circular symmetry in the eigenfunction coordinates). The network sizes can be found in [183].

Training: Similar as described in Box 4. (see [183] for the details).

Key findings:

- *SciNet* finds coordinates that linearize the dynamics of the nonlinear pendulum (see Fig. 10.2).
- The frequency ω decreases continuously in $r_1^2 + r_2^2$ as the energy of the pendulum is increased (see Fig. 10.2).

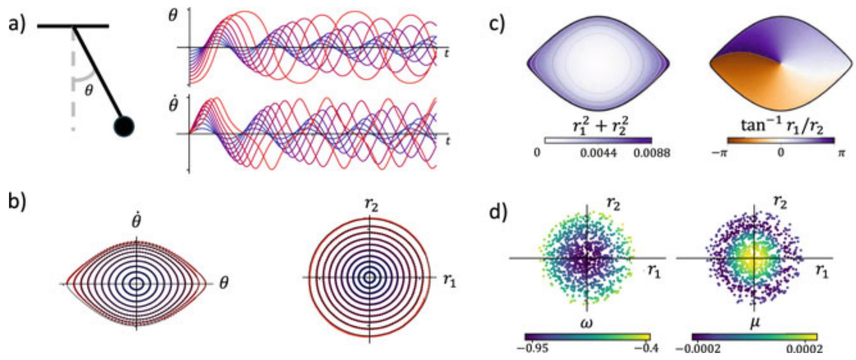


Fig. 10.2 Nonlinear pendulum (Figure reproduced from Lusch et al. *Nature Communications* 2018, [218]). The figure shows the results from [183], where *SciNet* is applied to observation data from a nonlinear pendulum (see Box 6 for the details). *SciNet* learns to accurately predict the time evolution of the nonlinear pendulum (Fig. 10.2a) and finds a coordinate transformation given by an encoder $E : (\theta, \dot{\theta}) \mapsto (r_1, r_2)$ that implements Koopman eigenfunctions (represented in Fig. 10.2c) and maps the angular displacement and velocity to coordinates (r_1, r_2) that are evolved in time Δt according to update rules of the form $u_{\gamma=(\mu,\omega)}(r_1, r_2) = \exp(\mu \Delta t)([\cos(\omega \Delta t) r_1 - \sin(\omega \Delta t) r_2], \sin(\omega \Delta t) r_1 + \cos(\omega \Delta t) r_2])$. Figure 10.2b shows *SciNet*'s prediction for θ and $\dot{\theta}$ and the corresponding values for r_1 and r_2 for ten initial conditions that are evolved in time until the relative prediction error reaches 10%. The dependence of the parameters μ and ω determining the update rule on r_1 and r_2 is shown in Fig. 10.2d. Note that the (absolute value of the) frequency ω decreases if the amplitude of the pendulum increases and that the damping factor μ stays approximately constant

The results are presented in Fig. 10.2 and taken from [183]. *SciNet* is able to predict the time evolution of the nonlinear pendulum with high accuracy. Furthermore, it finds a coordinate transformation from $(\theta, \dot{\theta})$ to (r_1, r_2), such that the representation (r_1, r_2) evolves according to linear dynamics. More concretely, the encoder implements Koopman eigenfunctions with eigenvalues that are allowed to continuously depend on $r_1^2 + r_2^2$. The variation in the eigenvalues allows to handle the continuous spectrum of the nonlinear pendulum in an elegant and interpretable way. As shown in Fig. 10.2, *SciNet* correctly identifies the dependence of the frequency ω on the energy of the pendulum.

10.4 Conservation of Angular Momentum

One of the most important concepts in physics is that of conservation laws, such as conservation of energy and angular momentum. While their relation to symmetries makes them interesting to physicists in their own right, conservation laws are also of practical importance. If two systems interact in a complex way, we can use conservation laws to predict the behaviour of one system from the behaviour of the other, without studying the details of their interaction. For certain types of questions, conserved quantities therefore act as a compressed representation of joint properties of several systems.

We consider the scattering experiment discussed in [22], shown in Fig. 10.3 and described in Box 7, where two point-like particles collide. Given the initial angular momentum of the two particles and the final trajectory of one of them, a physicist can predict the trajectory of the other using conservation of total angular momentum.

To see whether *SciNet* makes use of angular momentum conservation in the same way as a physicist would do, we train it with (simulated) experimental data as described in Box 7 with one latent neuron, and add Gaussian noise to show that the encoding and decoding are robust. Indeed, *SciNet* does exactly what a physicist would do and stores the total angular momentum in the latent representation (Fig. 10.3b). This example shows that *SciNet* can recover conservation laws, and suggests that they emerge naturally from compressing data and asking questions about joint properties of several systems.

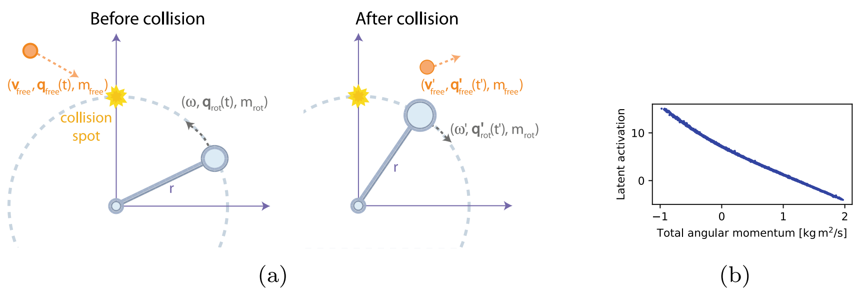

Before collision
After collision
$(\mathbf{v}_{free}, \mathbf{q}_{free}(t), m_{free})$
collision spot
$(\omega, \mathbf{q}_{rot}(t), m_{rot})$
r
$(\mathbf{v}'_{free}, \mathbf{q}'_{free}(t'), m_{free})$
$(\omega', \mathbf{q}'_{rot}(t'), m_{rot})$
Latent activation
10
0
−1
0
1
2
Total angular momentum [kg m²/s]
(a)
(b)

Fig. 10.3 Collision under conservation of angular momentum (Figure reproduced from Iten and Metger et al., *Physical Review Letters*, 2020 [22]). In a classical mechanics scenario where the total angular momentum is conserved, the neural network learns to store this quantity in the latent representation. **a** Physical setting. A body of mass m_{rot} is fixed on a rod of length r (and of negligible mass) and rotates around the origin with angular velocity ω. A free particle with velocity $\mathbf{v}_{\text{free}}$ and mass m_{free} collides with the rotating body at position $\mathbf{q} = (0, r)$. After the collision, the angular velocity of the rotating particle is ω' and the free particle is deflected with velocity $\mathbf{v}'_{\text{free}}$. **b** Representation learned by *SciNet*. Activation of the latent neuron as a function of the total angular momentum. *SciNet* learns to store the total angular momentum, a conserved quantity of the system

Box 7: Two-body collision with angular momentum conservation [22] (Sect. 10.4)

Problem: Predict the position of a particle fixed on a rod of radius r (rotating about the origin) after a collision at the point $(0, r)$ with a free particle (in two dimensions, see Fig. 10.3a).

Physical model: Given the total angular momentum before the collision and the velocity of the free particle after the collision, the position of the rotating particle at time t'_{pred} (after the collision) can be calculated from angular momentum conservation: $J = m_{\text{rot}} r^2 \omega - r m_{\text{free}} (\mathbf{v}_{\text{free}})_x = m_{\text{rot}} r^2 \omega' - r m_{\text{free}} (\mathbf{v}'_{\text{free}})_x = J'$.

Observation:

Time series of both particles before the collision: $o = [(t_i^{\text{rot}}, \mathbf{q}_{\text{rot}}(t_i^{\text{rot}})), (t_i^{\text{free}}, \mathbf{q}_{\text{free}}(t_i^{\text{free}}))]_{i \in \{1,\ldots,5\}}$, with times t_i^{rot} and t_i^{free} randomly chosen for each training sample. Masses $m_{\text{rot}} = m_{\text{free}} = 1\text{kg}$ and the orbital radius $r = 1\text{m}$ are fixed; initial angular velocity ω, initial velocity $\mathbf{v}_{\text{free}}$, the first component of $\mathbf{q}_{\text{free}}(0)$ and final velocity $\mathbf{v}'_{\text{free}}$ are varied between training samples. Gaussian noise ($\mu = 0, \sigma = 0.01\text{m}$) is added to all position inputs.

Question: Prediction time and position of free particle after collision: $q = \left(t'_{\text{pred}}, [t'_i, \mathbf{q}'_{\text{free}}(t'_i)]_{i \in \{1,\ldots,5\}}\right)$.

Correct answer: Position of rotating particle at time t'_{pred}: $a^\star = \mathbf{q}'_{\text{rot}}(t'_{\text{pred}})$.

Network structure: Network depicted in Fig. 9.1 with one latent neuron and specifications given in Table C.1.

Training: Described in Box 1 with parameters given in Table C.2.

Key findings:

- *SciNet* predicts the position of the rotating particle with root mean square prediction error below 4% (with respect to the radius $r = 1\text{m}$).
- *SciNet* is resistant to noise.
- *SciNet* stores the total angular momentum in the latent neuron.

10.5 Representation of Qubits

Quantum state tomography is an active area of research [219]. Ideally, we look for a *faithful representation* of the state of a quantum system, such as the wave function: a representation that stores all information necessary to predict the probabilities of the outcomes for arbitrary measurements on that system. However, to specify a faithful representation of a quantum system it is not necessary to perform all theoretically possible measurements on the system. If a set of measurements is sufficient to reconstruct the full quantum state, such a set is called *tomographically complete*.

Here we discuss the results from [22, 23] showing that, based only on (simulated) experimental data and without being given any assumptions about quantum theory, *SciNet* recovers a faithful representation of the state of small quantum systems and can make accurate predictions. In particular, this allows us to infer the dimension of the system and distinguish tomographically complete from incomplete measurement sets. Further, *SciNet* can separate parameters in the representation according to locality conditions (using the network structure to find operationally meaningful representations shown in Fig. 9.2) and end up with a similar representation of two qubits as given in [220]. Boxes 8 and 9 summarize the setting and the results.

A pure state on n qubits can be represented by a normalized complex vector $|\psi\rangle \in \mathbb{C}^{2^n}$, where two states $|\psi\rangle$ and $|\psi'\rangle$ are identified if and only if they differ by a global phase factor, i.e., if there exists $\phi \in \mathbb{R}$ such that $|\psi\rangle = e^{i\phi}|\psi'\rangle$. The normalization condition and irrelevance of the global phase factor decrease the number of free parameters of a quantum state by two. Since a complex number has two real parameters, a single-qubit state is described by $2 \times 2^1 - 2 = 2$ real parameters, and a state of two qubits is described by $2 \times 2^2 - 2 = 6$ real parameters.

Box 8: Representation of pure one- and two-qubit states [22] (Sect. 10.5.1)

Problem: Predict the measurement probabilities for any binary projective measurement $|\omega\rangle\langle\omega| \in \mathbb{C}^{2^n}$ on a pure n-qubit state $|\psi\rangle \in \mathbb{C}^{2^n}$ for $n = 1, 2$.
Physical model: The probability to measure 0 on the state $|\psi\rangle \in \mathbb{C}^{2^n}$ performing the measurement $|\omega\rangle\langle\omega| \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^n}$ is given by $p(\omega, \psi) = |\langle\omega\rangle\,\psi|^2$.
Observation: Operational parameterization of a state $|\psi\rangle$: $o = [p(\alpha_i, \psi)]_{i\in\{1,\dots,n_1\}}$ for a fixed set of random binary projective measurements $\mathcal{M} := \{|\alpha_1\rangle\langle\alpha_1|, \dots, |\alpha_{n_1}\rangle\langle\alpha_{n_1}|\}$ ($n_1 = 10$ for one qubit, $n_1 = 30$ for two qubits).
Question: Operational parameterization (see Remark 10.1) of a measurement $|\omega\rangle\langle\omega|$: $q = [p(\beta_i, \omega)]_{i\in\{1,\dots,n_2\}}$ by its outcome probabilities on a fixed set of random states $\mathcal{S} := \{|\beta_1\rangle, \dots, |\beta_{n_2}\rangle\}$ ($n_2 = 10$ for one qubit, $n_2 = 30$ for two qubits).
Correct answer: $a^\star(\omega, \psi) = p(\omega, \psi) = |\langle\omega\rangle\,\psi|^2$.
Network structure: Network depicted in Fig. 9.1 with varying numbers of latent neurons and specifications given in Table C.1.
Training: Described in Box 1 with parameters given in Table C.2.
Key findings:

- *SciNet* can be used to determine the minimal number of parameters necessary to describe the state $|\psi\rangle$ (see Fig. 10.4) without being provided with any prior knowledge about quantum physics.
- *SciNet* distinguishes tomographically complete and incomplete sets of measurements (see Fig. 10.4).

More generally, one may also consider mixed quantum states, which can be considered as a (classical) probabilistic mixture of pure quantum states. Formally, we describe mixed quantum states on n qubits by positive semi-definite matrixes $\rho \in \mathbb{C}^{2^n} \times \mathbb{C}^{2^n}$ with unit trace. Since there exists an eigendecomposition with real eigenvalues for any positive semi-definite matrix ρ, we can write $\rho = \sum_{i=1}^{2^n} p_i |\psi_i\rangle\langle\psi_i|$.[3] Since ρ has unit trace, we have $\sum_i p_i = 1$ and we can consider ρ as a probabilistic mixture of the pure states $|\psi_i\rangle$. A pure state $|\psi\rangle$ can then be considered as a special case of a density matrix $\rho = |\psi\rangle\langle\psi|$ with a single eigenvalue that is equal to one. For further details and a more in depth introduction into the formalism for quantum states we refer to [72].

[3] The notation $|\cdot\rangle$ and $\langle\cdot|$ is common in quantum mechanics and is called the "bra-ket" notation. The reader may consider the ket-vectors $|\cdot\rangle$ as a row vector, where a bra-vector $\langle\cdot|$ is considered as a column vector. Further, the entries from $\langle\psi|$ are the conjugate transposed entries of $|\psi\rangle$.

Here, we consider binary projective measurements on n qubits. These measurements are operators of the form $|\omega\rangle\langle\omega|$, which are fully described by a vector $|\omega\rangle \in \mathbb{C}^{2^n}$, with measurement outcomes labeled by 0 for the projection on $|\omega\rangle$ and 1 otherwise. The probability to get outcome 0 when measuring $|\omega\rangle\langle\omega|$ on a quantum system in a pure state $|\psi\rangle$ is then given by $p(\omega, \psi) = |\langle\omega\rangle\, \psi|^2$, where $\langle\cdot\rangle\,\cdot$ denotes the standard scalar product on $\mathbb{C}^{2^n}$. More generally, the probability to get outcome 0 when measuring ω on a quantum system in a mixed state ρ is given by $p(\omega, \rho) = \operatorname{tr} |\omega\rangle\langle\omega|\rho$. Note that this is consistent with the probability stated for pure states setting $\rho = |\psi\rangle\langle\psi|$.

To generate the training data for *SciNet*, we assume that an agent B has access to a preparation experiment consisting of two devices. The first device creates (many copies of) a quantum system in a state ρ, which depends on the setting of the dials and buttons of the device. The second device can perform any binary projective measurements in the set $\mathcal{M} := \left\{|\alpha_1\rangle\langle\alpha_1|, \ldots, \left|\alpha_{n_1}\right\rangle\left\langle\alpha_{n_1}\right|\right\}$, which we would like to use to determine the state of the quantum system. Agent B performs all measurements in $\mathcal{M}$ several times on the same quantum state ρ to estimate the probabilities $p(\alpha_i, \rho)$ of measuring 0 for the i-th measurement. These probabilities form the observation given to *SciNet*.

Based on a representation for the collected data by agent B, agents C_j are asked to predict the measurement outcomes of measurements $\left|\omega_j\right\rangle\left\langle\omega_j\right|$ chosen from a set $\mathcal{Q}^j$. To parameterize the measurement $\left|\omega_j\right\rangle\left\langle\omega_j\right|$ we choose sets of states

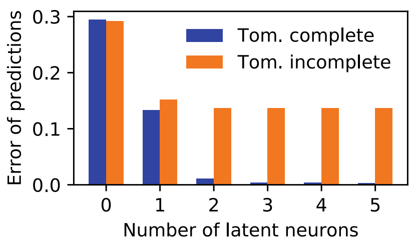
$$\mathcal{S}^j := \left\{\left|\beta_1^j\right\rangle, \ldots, \left|\beta_{n_2}^j\right\rangle\right\} .$$

The probabilities $p(\omega_j, \beta_1^j), \ldots, p(\omega_j, \beta_{n_2}^j)$ are provided to the agent C_j as the question input. The agent C_j then has to predict the probability $p(\omega_j, \rho)$ for measuring the outcome 0 on the state ρ when performing the measurement $\left|\omega_j\right\rangle\left\langle\omega_j\right|$.

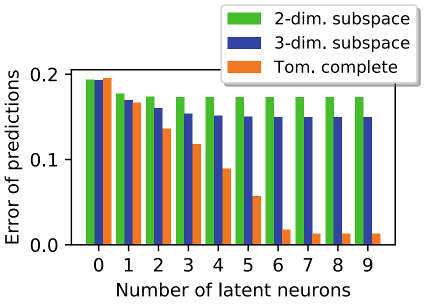
Remark 10.1 We could parameterize the set of possible measurements by any parameterization that is natural for the experimental setup, for example the settings of the dials and buttons on an experimental apparatus. Such a natural parameterization is assumed to fully specify the measurement, in the sense that the same settings on the experimental apparatus will always result in the same measurement being performed. Because $\mathcal{S}^j$ only represents our choice for parameterizing the measurement setup, it is natural to assume that $\mathcal{S}^j$ contains enough states to fully characterize all the measurements in $\mathcal{Q}^j$.

10.5.1 Minimal Representations for Pure Quantum States

As described in [22], we use *SciNet* in the quantum setting given above to find the required number of real parameters to describe pure quantum states on one and two qubits. Further, we then demonstrate how to use *SciNet* to determine if a set



(a) One qubit.



(b) Two qubits.

Fig. 10.4 Quantum tomography (Figure taken from Iten and Metger et al., *Physical Review Letters*, 2020 [22]). *SciNet* is given tomographic data for one or two qubits and an operational description of a measurement as a question input and has to predict the probabilities of outcomes for this measurement. We train *SciNet* with both tomographically complete and incomplete sets of measurements, and find that, given tomographically complete data, *SciNet* can be used to find the minimal number of parameters needed to describe a quantum state (two parameters for one qubit and six parameters for two qubits). Tomographically incomplete data can be recognized, since *SciNet* cannot achieve perfect prediction accuracy in this case, and the prediction accuracy can serve as an estimate for the amount of information provided by the tomographically incomplete set. The plots show the root mean square error of *SciNet's* measurement predictions for test data as a function of the number of latent neurons

of measurements is tomographically complete or not. More concretely, we use the network structure for *SciNet* shown in Fig. 9.1 and the training process described in Box 1 to find the minimal number of required parameters to represent quantum states. If the prediction accuracy stays low for any number of latent neurons, we can conclude that the provided observation data does not contain sufficient information, which means in the considered case that the set of measurements $\mathcal{M}$ is tomographically incomplete. How to operationally separate the parameters in the representation is then considered in Sect. 10.5.2.

We consider an agent B performing measurements on pure quantum states $|\psi\rangle$ and one agent $C := C_1$ who is asked to predict the measurement probability $p(\omega, \psi)$ of measuring zero when performing the measurement $|\omega\rangle\langle\omega|$ on $|\psi\rangle$. We train *SciNet* with different pairs $(|\omega\rangle\langle\omega|, |\psi\rangle)$ for one and two qubits, keeping the measurement sets $\mathcal{M}$ and $\mathcal{S} := \mathcal{S}^1$ fixed for each choice of the qubit number. We choose $n_1 = n_2 = 10$ for the single-qubit case and $n_1 = n_2 = 30$ for the two-qubit case. The setting is summarized in Box 8.

The results are shown in Fig. 10.4. Varying the number of latent neurons, we can observe how the quality of the predictions improves as we allow for more parameters in the representation of $|\psi\rangle$. To minimize statistical fluctuations due to the randomized initialization of the network, each network specification is trained three times and the run with the lowest mean square prediction error on the test data is used.

For the cases where $\mathcal{M}$ is tomographically complete, the plots in Fig. 10.4 show a drop in prediction error when the number of latent neurons is increased up to two or six for the cases of one and two qubits, respectively.[4] This is in accordance with the number of parameters required to describe a one- or a two-qubit state. Thus, *SciNet* allows us to extract the dimension of the underlying quantum system from tomographically complete measurement data, without any prior information about quantum mechanics.

SciNet can also be used to determine whether the measurement set $\mathcal{M}$ is tomographically complete or not. To generate tomographically incomplete data, we choose the measurements in $\mathcal{M}$ randomly from a subset of all binary projective measurements. Specifically, the quantum states corresponding to measurements in $\mathcal{M}$ are restricted to random *real* linear superpositions of k orthogonal states, i.e., to a (real) k-dimensional subspace. For a single qubit, we use a two-dimensional subspace; for two quibts, we consider both two- and three-dimensional subspaces.

Given tomographically incomplete data about a state $|\psi\rangle$, it is not possible for *SciNet* to predict the outcome of the final measurement perfectly regardless of the number of latent neurons, in contrast to the tomographically complete case (see Fig. 10.4). Hence, we can deduce from *SciNet's* output that $\mathcal{M}$ is an incomplete set of measurements. Furthermore, this analysis provides a qualitative measure for the amount of information provided by the tomographically incomplete measurements: in the two-qubit case, increasing the subspace dimension from two to three leads to higher prediction accuracy and the required number of latent neurons increases.

Remark 10.2 (Efficient representation of states of many body systems) Quantum many body systems are challenging to describe because of the exponential complexity of the many-body wave function encoding the non-trivial correlations between the particles. In [222–236] neural networks are used to efficiently represent states of quantum many body systems, to find their ground states and even to simulate their unitary time evolution. In particular, in [223], variational autoencoders are used to approximate the distribution of the measurement outcomes of a specific quantum state for a fixed measurement basis and the size of the neural network can provide an estimate for the complexity of the state. In contrast, the approach presented in this section does not focus on an efficient representation of a given quantum state and it is not specifically designed for learning representations of quantum systems. Nevertheless, *SciNet* can be used to produce representations of arbitrary states of simple quantum systems without retraining.

[4] In the case of a single qubit, there is an additional small improvement in going from two to three latent neurons: this is a technical issue caused by the fact that any two-parameter representation of a single qubit, for example the Bloch sphere representation, includes a *cyclic parameter*, which cannot be exactly represented by a continuous encoder (see Appendix D). The same likely applies in the case of two qubits, going from 6 to 7 latent neurons. This restriction also makes it difficult to interpret the details of the learned representation. Recent work [221] tackles this problem by using theory of manifolds and splitting manifolds into parts for which there is continuous encoding. Although this work is technically very useful, it is not discussed in detail here because it does not contribute a lot to the discussion at the conceptual level.

10.5.2 Local Representation of Two-Qubit States

In the last section, we investigated the number of parameters required to describe (pure) quantum states. Let us now follow the approach in [23, 237] to separate the parameters for a representation of a (mixed) two-qubit state ρ in an operationally meaningful way, in the sense as described in Sect. 8.4.2.2. More concretely, we use the network structure for *SciNet* shown in Fig. 9.2 and the training process described in Box 3.

Box 9: Local representation of two-qubit states [23] (Sect. 10.5.2)

Problem: Three agents C_1, C_2 and C_3 have to predict the measurement probabilities for binary projective measurements on two qubits. Thereby, agent C_1 and C_2 are asked questions about measurement output probabilities on the first and second qubit, and gent C_3 is asked to predict joint measurement output probabilities on both qubits.

Physical model: The probability to measure 0 on the state ρ performing the measurement $|\omega\rangle\langle\omega|$ is given by $p(\omega, \rho) = \mathrm{tr}\, |\omega\rangle\langle\omega|\, \rho$.

Observation: Operational parameterization of a state ρ: $o = [p(\alpha_i, \rho)]_{i\in\{1,\dots,n_1\}}$ for a fixed set of random binary projective measurements $\mathcal{M} := \{|\alpha_1\rangle\langle\alpha_1|, \dots, |\alpha_{75}\rangle\langle\alpha_{75}|\}$.

Questions: Operational parameterization (see Remark 10.1) of measurements $\left|\omega_j\right\rangle\left\langle\omega_j\right|$: $q^j = \left[p(\beta_i^j, \omega_j)\right]_{i\in\{1,\dots,75\}}$ by their outcome probabilities for fixed sets of random states

$$\mathcal{S}^j := \left\{\left|\beta_1^j\right\rangle, \dots, \left|\beta_{75}^j\right\rangle\right\} ,$$

where $\{\left|\beta_i^j\right\rangle\}_{i\in\{1,\dots,75\}}$ are chosen to be of the form $|\beta_i^1\rangle = \left|\tilde{\beta}_i^1\right\rangle \otimes |0\rangle$, $|\beta_i^2\rangle = |0\rangle \otimes \left|\tilde{\beta}_i^2\right\rangle$ for random one-qubit states $\left|\tilde{\beta}_i^1\right\rangle$ and $\left|\tilde{\beta}_i^2\right\rangle$, and where $|\beta_i^3\rangle$ are random two-qubit states. The measurements are chosen to be of the following form: $|\omega_1\rangle\langle\omega_1| = |\psi_1\rangle\langle\psi_1| \otimes \mathrm{id}$, $|\omega_2\rangle\langle\omega_2| = \mathrm{id} \otimes |\psi_2\rangle\langle\psi_2|$ and $|\omega_3\rangle\langle\omega_3| = |\psi_3\rangle\langle\psi_3|$, where $|\psi_1\rangle$ and $|\psi_2\rangle$ are one-qubit states, and $|\psi_3\rangle$ is a two-qubit state.

Correct answers: The decoder D_j should output $a^\star(\omega_j, \rho) = p(\omega_j, \rho) = \mathrm{tr}\left|\omega_j\right\rangle\left\langle\omega_j\right|\rho$.

Network structure: Network depicted in Fig. 9.2 with 15 latent neurons[a] and three decoders and network sizes given in Table C.1.

Training: Described in Box 3 with parameters given in Table C.2.

Key finding: *SciNet* finds a representation for two-qubit states that separates parameters storing local degrees of freedom of each of the qubits. The found representation is similar to the one presented in [220].

[a] For simplicity, we have choosen the minimal number of required latent neurons here. This number could be found by successively increasing the number of latent neurons until *SciNet* achieves hight prediction accuracy.

For the preparation experiments, an agent B fixes 75 randomly chosen binary measurements $\mathcal{M} := \{|\alpha_1\rangle\langle\alpha_1|, \ldots, |\alpha_{75}\rangle\langle\alpha_{75}|\}$ that are performed on two-qubit states ρ that are varied during the training. Three agents C_1, C_2 and C_3 are now required to answer different questions about the two-qubit system:

- Agent C_1 and C_2 are asked questions about measurement output probabilities on the first and second qubit, respectively.
- Agent C_3 is asked to predict joint measurement output probabilities on both qubits.

The question inputs consisting of a binary measurement $|\omega_j\rangle\langle\omega_j|$ (on one or two qubits, respectively) are parametrised by 75 randomly chosen states $\mathcal{S}^j := \left\{\left|\beta_1^j\right\rangle, \ldots, \left|\beta_{75}^j\right\rangle\right\}$. The details about the setting are summarized in Box 9.

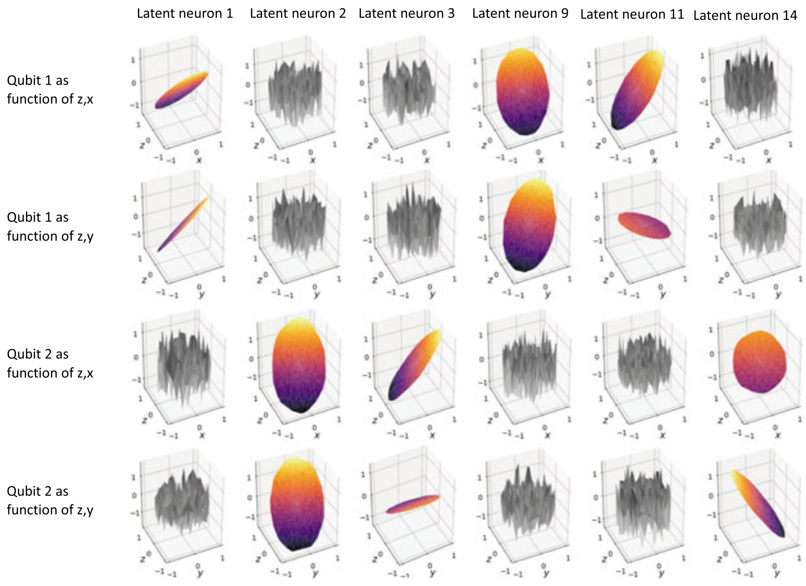


Fig. 10.5 Results for the local representation of two-qubit states (Figure taken from Nautrup, Metger and Iten et al., 2020 [23]). We consider a quantum-mechanical system of two qubits. An encoder maps tomographic data of a two-qubit state to a representation of the state. Three agents C_1, C_2 and C_3 are asked questions about the measurement output probabilities on the two-qubit system, where a question is given as the parameterisation of a measurement. Agents C_1 and C_2 are asked to predict measurement outcome probabilities on the first and second qubit, respectively. The third agent C_3 is tasked to predict measurement probabilities for arbitrary measurements on the full two-qubit system. In total, 15 latent neurons are required to answer the questions of all agents C_1, C_2 and C_3. Agent C_3 requires access to all parameters, while agents C_1 and C_2 need only access to two disjoint sets of three parameters, encoded in latent neurons 1,9,11 and 2,3,14 respectively. The plots show the activation values for these latent neurons in response to changes in the local degrees of freedom of each qubit, with the bottom axes of the plots denoting the components of the reduced one-qubit state $\rho = 1/2(\mathrm{id} + x\,\sigma_x + y\,\sigma_y + z\,\sigma_z)$ on either qubit 1 or 2

We find that three latent neurons are used for each of the local qubit representations as required by agents C_1 and C_2. These local representations store combinations of the x-,y- and z-component of the Bloch sphere representation $\rho = 1/2(\mathrm{id} + x\sigma_x + y\sigma_y + z\sigma_z)$ of a singe qubit (see Fig. 10.5), where σ_x, σ_y, σ_z denote the Pauli matrices. In general, a two-qubit mixed state ρ is described by 15 parameters, since a Hermitian 4×4 matrix is described by 16 parameters, and one parameter is determined by the others due to the unit trace condition. Indeed, we find that the agent who has to predict the outcomes of the joint measurements accesses all 15 latent neurons, including the ones storing the two local representations. These numbers correspond to the numbers found in the analytical approach in [220].

10.6 Charged Particles

In this section, we investigate a toy example from classical mechanics that is considered in [23, 237] to demonstrate that the criterium for a operationally meaningful separation of parameters (Sect. 8.4.2.2) leads to a discovery of the same parameters that we know from physical text books. In contract to the example discussed in Sect. 10.2, the methods used in this section are not biased by the distribution over the training data. We use the network structure for *SciNet* shown in Fig. 9.2 and the training process described in Box 3. Further, we demonstrate how to use several encoding agents instead of one by means of the same example.

We consider the setup shown in Fig. 10.6 consisting of two particles $p_1 = (m_1, q_1)$ and $p_2 = (m_2, q_2)$ with masses m_1, m_2 and charges q_1, q_2. Experimenters (agents B_i) can perform the following reference experiments to gain information about the properties of the particles:

1. The experimenter can elastically collide each of the particles p_i, initially at rest, with an uncharged reference mass m_{ref} moving at a fixed reference velocity v_{ref}. Then, the agent observes a time series of positions of the particle p_i after the collision.
2. We place a particle p_i at the origin of a coordinate system at rest, and place a reference particle $p_{\mathrm{ref}} = (m_{\mathrm{ref}}, q_{\mathrm{ref}})$ with fixed mass and charge at a fixed distance d_0 to the origin. Both particles are free to move. We observe a time series of positions of the particle p_i as it moves due to the Coulomb interaction between itself and the reference particle.

We choose natural questions asked to different decoding agents C_j (see Fig. 8.3). In the physical setting about which we ask the questions, the initial positions of the particles p_i and the positions of two target holes h_i are fixed.

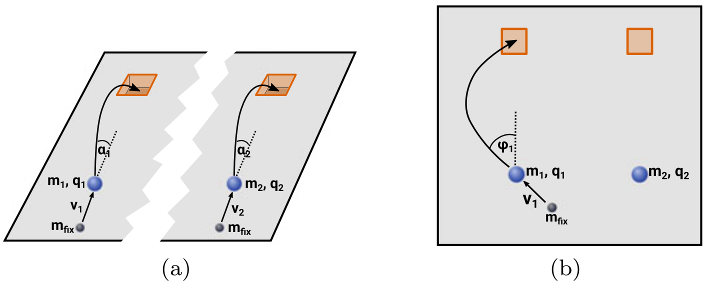


Fig. 10.6 Toy example with charged masses (Figure taken from Nautrup, Metger and Iten et al., 2020 [23, 237]). **a** There are two separated decoding agents C_1 and C_2, so there is no interaction between their individual experimental setups. Each agent is required to *shoot* a particle p_i (of mass m_i and charge q_i) into a hole h_i in the presence of a fixed gravitational field. They do this by elastically colliding a projectile of fixed mass m_{fix} with the particle p_i. The distance to the hole is fixed. Challenging agents choose the velocity v_i with which the projectile is shot, and provide this information to the decoding agents as a question input. The correct answer is the angle α_i out of the plane of the ground, such that if the projectile is fired with the given velocity at this angle, the particle p_i lands directly in the hole. **b** Now we consider two decoding agents C_3 and C_4 in a setting where the two particles are subject to the Coulomb interaction between each other. The agents C_3 and C_4 are again required to shoot a projectile at the particles p_1 and p_2, respectively. The charged particle will move in the Coulomb field of the other agent's particle. Similar to the situation in a), the agents are given velocities v_3 or v_4 as a questions and they have to predict the angles φ_1 or φ_2 in the plane of the ground, respectively, such that if the mass is fired at this angle, it will "roll" on the frictionless ground into the hole while the position of the other charge stays fixed. (The experiment is then repeated with the roles of the agents reversed, i.e., the agent that first fired his mass now fixes it at its starting position, and vice versa)

- Each decoding agent C_1 and C_2 are given projectiles with a fixed mass m_{fix} and a challenging agent chooses a velocity v_j with which the projectile will hit the particle p_j. The decoding agent C_j (described by a decoder D_j) is then asked to predict the correct angle α_j in the yz-plane with which the agent shoots his projectile against the particle p_j, such that the particle p_j lands directly, i.e., without bouncing, in its target hole h_j (under the influence of a constant homogeneous gravitational field).
- Similarly, agents C_3 and C_4 are also given projectiles, whose chosen velocities v_3 and v_4 are given as a question input to decoders D_3 and D_4. The goal of the decoders is to predict the angle φ_1 and φ_2 in the xy-plane so that the particle p_1 and p_2 will fly into its target hole, under the influence of the Coulomb field of the other particle, which stays fixed.

In both cases, we restrict the possible velocities of the projectiles such that there exists a (unique) angle that makes the particle land in the hole. Further, the prediction loss is given by the squared difference between the angle chosen by the agent and the *correct* angle that would have landed the mass directly into the hole; this correct angle can be determined by experiments on the system.

Box 10: Charged particles [23] (Sect. 10.6)

Problem: There are two physical challenges shown in Fig. 10.6 involving two charged particles $p_i = (m_i, q_i)$ with masses m_i and charges q_i. In challenge a), two decoding agents C_1 and C_2 have to predict the shooting-angles α_1 and α_2 in yz-plane for a given schooting-velocity, such that the particle p_1 or p_2, respectively, land in a target hole under the influence of a gravitational field. Similarly, in challenge b), two decoding agents C_3 and C_4 have to predict the shooting-angles φ_1 and φ_2 in the xy-plane for a given shooting-velocity, such that the particles p_1 or p_2, respectively, lands in a target hole.

Physical model: a) A particle with mass m_i moves in a constant gravitational field with strength g in negative z-direction. b) A charged particle p_i moves according to the action of the Coulomb force of the other particle. For a distance r between the particles, the Coulomb force is proportional to $q_1 q_2 / r^2$.

Observations: See the experiments 1 and 2, described at the start of Sect. 10.6. In the case of two encoding agents E_i, each of them has only access to the experiments related to particle p_i.

Questions: In a) and b) the questions are given by the velocity with which the projectile will hit the particle, i.e., $q^j = v_j$ for $j \in \{1, 2, 3, 4\}$.

Correct answers: a) Assuming that the target is placed at $x = d_0$, we find that $\alpha_i = \frac{1}{2} \arcsin \frac{d_0 g}{v_i^2}$. b) The answers in this setting are discussed in Appendix E.

Network structure: **Single encoder:** Network depicted in Fig. 9.2 with one encoder, 3 latent neurons[a] and four decoders. **Multiple encoders:** Network depicted in Fig. 9.2 with two encoders, 4 latent neurons[a] and four decoders. For both cases, the network sizes are given in Table C.1.

Training: Described in Box 3 with parameters given in Table C.2.

Key finding: The key findings for a single and for multiple encoders are:

- **Single encoder:** *SciNet* stores the masses m_1 and m_2 in separate latent neurons (Fig. 10.7). The third latent neuron stores the product of the charges $q_1 q_2$, necessary to quantify the strength of the Coulomb interaction. Crucially, the separation is not biased by the distribution over the training data.
- **Multiple encoders:** There is no way for the encoding agents to directly encode the product of the charges $q_1 \cdot q_2$. Instead, the representation produced by each encoding agent stores q_i individually.

[a] For simplicity, we have choosen the minimal number of required latent neurons here. This number could be found by successively increasing the number of latent neurons until *SciNet* achieves hight prediction accuracy.

10.6.1 Single Encoder

Let us first consider a single encoding agent B, and four decoding agents described by decoders D_1, D_2, D_3, D_4. The setting and the results are summarized in Box 10.

We use the training process described in Box 3 to train *SciNet*. To analyse the learnt representation, we plot the activation of the latent neurons for different examples

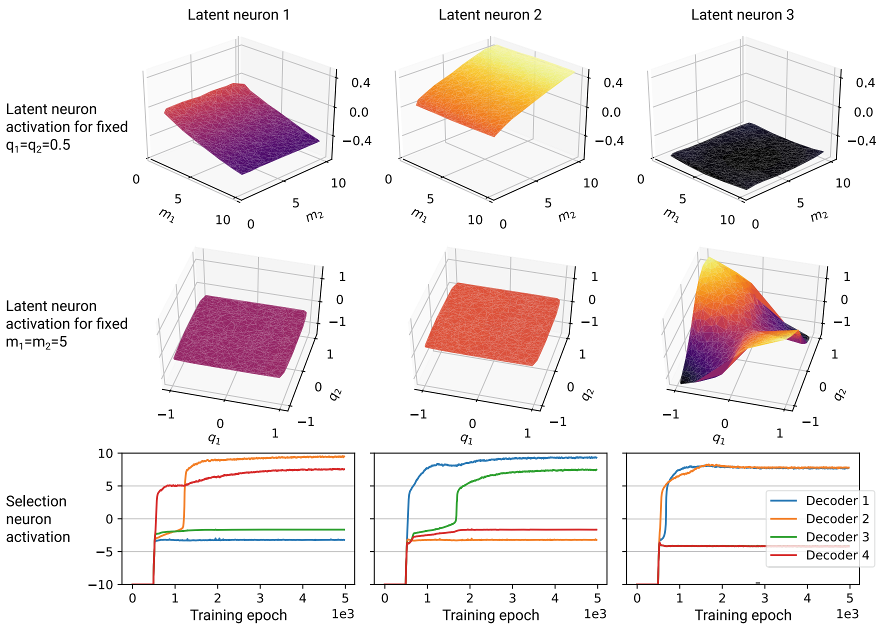


Fig. 10.7 Results for the example with charged masses (single encoder) (Figure taken from Nautrup, Metger and Iten et al., 2020 [23]). The used network (Fig. 9.2) has 3 latent neurons and each column of plots corresponds to one latent neuron. For the first row we generated input data with fixed charges $q_1 = q_2 = 0.5$ (ignoring the units) and variable masses m_1, m_2 in order to plot the activation of latent neurons as a function of the masses. We observe that latent neuron 1 and 2 store the masses m_1, m_2 respectively while latent neuron 3 remains constant. In the second row, we plot the neurons' activation in response to q_1, q_2 with fixed masses $m_1, m_2 = 5$. Here, the third latent neuron approximately stores $q_1 \cdot q_2$, which is the relevant quantity for the Coulomb interaction while the other neurons are independent of the charges.The third row shows which decoder receives information from the respective latent neuron. The y-axis corresponds to $\log(\sigma_i^j)$ for the i-th latent neuron and decoder j. Hence, the y-axis quantifies how much information of the latent neuron is transmitted by the 4 filters to the associated decoder as a function of the training epoch. Positive values mean that the filter does not transmit any information. Decoders 1 and 2 perform non-interacting experiments with particles (m_1, q_1) and (m_2, q_2), respectively. Decoders 3 and 4 perform the corresponding interaction experiments. As expected, we observe that the information about m_1 (latent neuron 1) is received by decoders 1 and 3 and the information about m_2 (latent neuron 2) is used by decoders 2 and 4. Since decoders 3 and 4 answer questions about interaction experiments, the product of charges (latent neuron 3) is received only by them (the green line of decoder 3 in the last plot is hidden below the red one)

with different (known) values of the masses m_1, m_2 and the charges q_1, q_2 against those known values. This corresponds to comparing the learnt representation to a hypothesised representation that we might already have. The plots are shown in Fig. 10.7. The first and second latent neurons are linear in m_1 and m_2, respectively, and independent of the charges; the third latent neuron has an activation that resembles the function $q_1 \cdot q_2$ and is independent of the masses. This means that the first and third latent neurons store the masses individually, as would be expected since the setup in Fig. 10.6a only requires individual masses and no charges. The third neuron roughly stores the product of the charges, i.e., the quantity relevant for the strength of the Coulomb interaction between the charges. This is used by the agents dealing with the setup in Fig. 10.6b, where the particle's trajectory depends on the Coulomb interaction with the other particle.

10.6.2 Multiple Encoders

One can easily generalize the above example to one where one uses several encoding agents. Instead of having a single agent B, we use two agents B_1 and B_2, where each of the agents has only access to one of the charged particles p_1 and p_2, i.e., agent B_i only observes the results of the experiment 10.6 associated with particle p_i. The two encoding agents then have to separately construct a representation of their observations. Then, the two representations are concatenated and treated like in the single-encoder setup; that is, for each decoder, a filter is applied to the concatenated

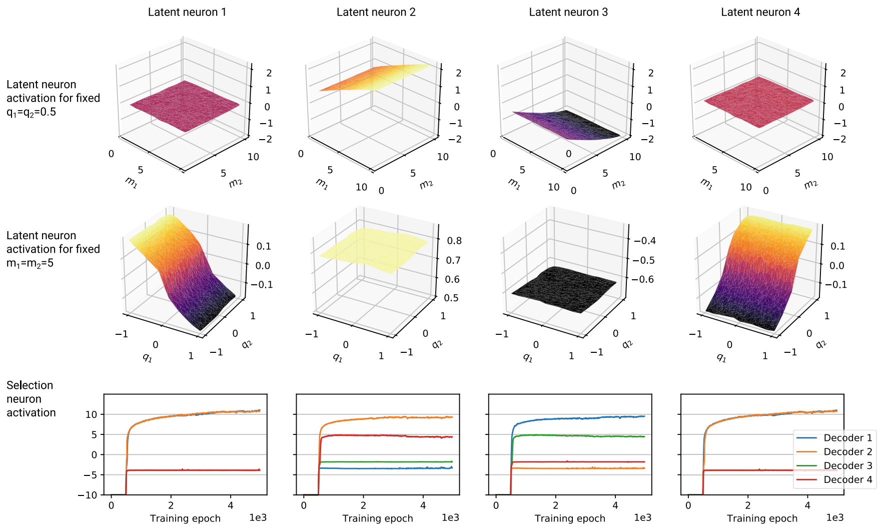


Fig. 10.8 Results for the example with charged masses (two encoders) (Figure taken from Nautrup, Metger and Iten et al., 2020 [23]). The used network (Fig. 9.2) has 4 latent neurons and each column of plots corresponds to one latent neuron. For an explanation of how these plots are generated, see the caption of Fig. 10.7. We observe that latent neurons 2 and 3 store the masses m_1 and m_2, respectively, while latent neurons 1 and 4 are independent of the mass. Latent neurons 1 and 4 store (a monotonic function of) the charges q_1 and q_2, respectively, and are indepependent of m_1 and m_2. The third row shows that the charges q_1 and q_2 are only transmitted to decoders 3 and 4, which are asked to make predictions about interaction experiments (the blue line of decoder 1 and the green line of decoder 3 are hidden under the orange and red lines, respectively, in both of these plots). The mass m_1, stored in the latent neuron 2, is transmitted to decoders 1 and 3, which are the two decoders that make predictions about particle p_1. Analogously, m_2 is transmitted to decoders 2 and 4, which make predictions about particle p_2

representation and the filtered representation is used as input for the decoder. The resulting network structure has similarity to the one in [238] used for summarizing text documents.

The results for this case are shown in Fig. 10.8 and summarized in Box 10. Comparing this result with the single-encoder case, we observe that here, the charges q_1 and q_2 are stored individually in the latent representation, whereas the single encoder stored the product $q_1 \cdot q_2$. This is because, even though the decoders still only require the product $q_1 \cdot q_2$, no single encoder has sufficient information to output this product: the inputs of encoders E_1 and E_2 only contain information about the individual charges q_1 and q_2, respectively, but not their product. Hence, the additional structure imposed by splitting the input among two encoders yields a representation with more structure, i.e., with the two charges stored separately. However the concatenated representation consists of four instead of three latent neurons as in the singe-encoder case, it is still minimal according to Definition 8.8. Indeed, one can see that the chosen structure with two encoders does not allow for a representation with only three latent neurons: If such a representation would exist, one encoder could only output one parameter (and the other encoder outputs two parameters), which is not enough to store the information about the mass and the charge.

10.7 Heliocentric Solar System

In the 16th century, Copernicus used observations of the positions of different planets on the night sky (Fig. 10.9a) to hypothesize that the Sun, and not the Earth, is in the centre of our solar system. This heliocentric view was confirmed by Kepler at the start of the 17th century based on astronomic data collected by Brahe, showing that the planets move around the Sun in elliptical orbits. Here, based on the work in [22], we show that *SciNet* similarly uses heliocentric angles (more exactly, the

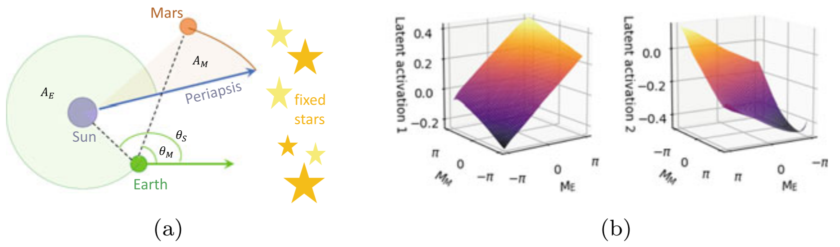


Fig. 10.9 Heliocentric model of the solar system. *SciNet* is given the angles of the Sun and Mars as seen from Earth at an initial time t_0 and has to predict these angles for later times. **a** Physical setting. The angles θ_S and θ_M of the Sun and Mars are observed from Earth (relative to the fixed star background). The areas A_E and A_M are swept out by a line segment joining Earth or Mars and the Sun, respectively. These areas are proportional to the mean anomalies $M_E \propto A_E$ and $M_M \propto A_M$ of Earth and Mars, respectively (see (10.1)). **b** Representation learned by *SciNet*. The activations $r_{1,2}$ of the two latent neurons at time t_0 (see Fig. 9.3) are plotted as a function of the mean anomalies M_E and M_M. The plots show that the network stores and evolves parameters that are linear combinations of these heliocentric angles.

mean anomalies) when forced to find a representation for which the time evolution of the variables takes a very simple form, a typical requirement for time-dependent variables in physics.

The observations given to *SciNet* (with the network structure shown in Fig. 9.3 with two latent neurons) are angles $\theta_M(t_0)$ of Mars and $\theta_S(t_0)$ of the Sun as seen from Earth at a starting time t_0 (which is varied during training). The time evolution network $u : \mathbb{R}^2 \mapsto \mathbb{R}^2$ is restricted to addition of a constant (the value of which is learned during training). Hence, we can smoothly parametrize the set of update rules by two parameters $\gamma = (\gamma_1, \gamma_2)$ as $u_\gamma(r_1, r_2) = (r_1 + \gamma_1, r_2 + \gamma_2)$. At each time step i, *SciNet* is asked to predict the angles as seen from Earth at the time $t_i = t_0 + i\Delta t$ based only on its representation at step i. The time step Δt is fixed and is chosen to be 25 d in our implementation. Because the question $\tilde{q}$ is the same at each time step, we do not need to feed it to the decoder explicitly.

Restricting to linear time evolution (by only allowing to add constants γ_i in the update rule) may look like a limitation that does only apply to very particular physical systems. However, such a restriction is actually equivalent to the assumption that the energy of the considered system is conserved. In such a case, there is a variable transformation such that the new canonical coordinates (called *action angles*) evolve linearly in time (see for example [239]).

We simulate the elliptical orbits of Mars and Earth around the sun (note that circular orbits were used in the original work [22]) and train *SciNet* with randomly chosen sequences of observations for around 50 steps of 25 d. Figure 10.9b shows the learned representation and confirms that *SciNet* indeed stores *heliocentric angles*. Indeed, *SciNet* stores a linear combination[5] of the mean anomalies $M_E \in [0, 2\pi)$ and $M_M \in [0, 2\pi)$ of Earth and Mars, i.e., 2π times the fractions of the elliptical orbits' periods that have elapsed since the planets passed their periapsis, the points in the orbits of the planets at which they are nearest to the Sun. The mean anomalies are chosen to lie in the interval $[0, 2\pi)$ because they can be interpreted geometrically as the angles that would be observed from the Sun in the case where Earth and Mars would follow circular orbits (with the same period) around the Sun. An alternative geometrical relation is that the areas A_E and A_M swept out by a line segment joining Earth or Mars and the Sun are proportional to the mean anomalies of Earth and Mars, respectively (see Fig. 10.9). More concretely, we have

$$A_E = \frac{M_E}{2\pi} A_E^{\text{tot}} \text{ and } A_M = \frac{M_M}{2\pi} A_M^{\text{tot}} , \tag{10.1}$$

where A_E^{tot} and A_M^{tot} denote the full area enclosed by the orbit of Earth and Mars, respectively. Hence, an alternative (and physically equivalent) interpretation is that

[5] If desired, one could disentangle the mean anomalies of Earth and Mars by using the methods described in Sect. 9.3.2. Indeed, we could introduce two decoding agents. The first should predict the angle θ_S of the Sun, and the second the angle of the Mars θ_M (as seen from Earth). To solve the tasks perfectly, the first agent requires the information about the mean anomaly of Earth and the second agent requires both, the mean anomaly of Earth and the Mars. Hence, the corresponding operationally meaningful representation would naturally separate the two mean anomalies.

SciNet stores linear combination of the areas swept out by the two planets in its latent neurons. The fact that the swept out areas grow linearly in time is stated in Kepler's second law of planet motion. We stress that the training data only contains angles observed from Earth, but *SciNet* nonetheless switches to a heliocentric representation. The details are given in Box 11.

Box 11: Heliocentric model of the solar system (Sect. 10.7)

Problem: Predict the angles $\theta_M(t)$ and $\theta_S(t)$ of Mars and the Sun as seen from Earth, given initial states $\theta_M(t_0)$ and $\theta_S(t_0)$.

Physical model: Earth and Mars orbit the Sun on elliptical orbits according to Kepler's laws of planet motion. The orbit of Earth is approximately a circle (eccentricity: 0.017) and the orbital velocity is approximately constant. The elliptical orbit of Mars has an eccentricity of 0.093 and the orbital velocity varies up to $\pm 10\%$ with respect to the mean orbital velocity.

Observation: Randomly chosen initial angles of Mars and the Sun as seen from Earth: $o = (\theta_M(t_0), \theta_S(t_0))$.

Question: Implicit.

Correct answer: Time series $\left[a_0, \ldots, a_n\right] = \left[(\theta_M(t_0), \theta_S(t_0)), \ldots, (\theta_M(t_n), \theta_S(t_n))\right]$ of $n = 20$ (later in training: $n = 50$) observations, with fixed time steps $\Delta t = t_{i+1} - t_i$ of 25 days.

Network structure: Network depicted in Fig. 9.3 with two latent neurons and allowing for time updates of the form $u_\gamma(r_1, r_2) = (r_1 + \gamma_1, r_2 + \gamma_2)$. The network sizes are given in Table C.1.

Training: As described in Box 4 with parameters given in Table C.2.

Key findings:

- *SciNet* predicts the angles of Mars and the Sun with a root mean square error below 0.5% (with respect to 2π).
- *SciNet* stores the mean anomalies M_E and M_M of the Earth and Mars with respect to the Sun in the two latent neurons (see Fig. 10.9b).

10.8 Several Particles Connected by Springs

In this section, we consider a physical systems consisting of many interacting objects as an example to demonstrate how physical prior knowledge about the object-structure can help to interpret *SciNet*'s representation (Sect. 8.6).

More concretely, we consider $k \in \{5, 10\}$ particles in a 2-dimensional box with no external forces, apart from the elastic collisions with the walls of the box [184]. Some particles are connected by a string with a fixed spring constant k, and hence, a force $F_{i,j} = k \left\| p_i - p_j \right\|_2$ acts between two particles i and j with positions $p_i \in \mathbb{R}^2$ and $p_j \in \mathbb{R}^2$ that are connected by such a spring. The details and the sample strategy are described in Box 12. The observation input to *SciNet* (with the structure given in Fig. 9.5 using GNNs) is given as time sequences of the 2-dimensional positions and velocities of the particles. The one-hot embeddings $h_{i,j} \in \mathbb{R}^2$ of the edges of the GNN used for the encoder can encode two different types of interactions. The

GNN is then evolved for $L = 2$ steps and representations $r_{i,j}$ for the interaction types between any objects i and j are sampled.

The decoder again consists of a GNN and takes the representation r as an input together with a question q that is described by an initial state of the positions and velocities of the particles at time t' as well as a number $M \in \mathbb{N}$ describing the number of time steps that the state of the particles should be evolved by for a time difference Δt. The decoder then has to predict the future states of the particles at a time $t' + M\Delta t$.

SciNet learns to predict the paths of the particles and the type of interactions between the particles with high accuracy [184]. It finds that there are two types of interactions between two particles, namely "no interaction" and "connected by a spring". This information can be extracted from the one-hot encodings that *SciNet* stores in its representation r.

Box 12: Several particles connected by springs [184] (Sect. 10.8)

Problem: Predict the time evolution of the positions and velocities of $k \in \{5, 10\}$ particles in a 2-dimensional box.

Physical model: The force on particle i at position p_i is given by the sum of the spring-forces $F_{i,j} = k \left\| p_i - p_j \right\|_2$ of the particles j at position p_j that are connected by a spring with particle i. Further, the particles collide elastically with the walls of the box. The trajectories of the particles are then simulated by solving Newton's equation with leapfrog integration [184].

Observation: Time sequences of position and velocities of all particles starting from a randomly chosen initial state: Randomly chosen initial positions p_i, where both components of p_i are samples from a Gaussian distribution $\mathcal{N}(0, 0.5)$ with mean equal to 0 and standard derivation of 0.5. The velocities of the particle are sampled uniformly from the set of 2-dimensional vectors with norm 0.5. A spring is placed between two particles i and $j \neq i$ with probability 50%.

Question: $q = [x_1(t'), \ldots, x_k(t'), M]$ with $k \in \{5, 10\}$, $M \in \mathbb{N}$ and states $x_i(t') = [p_i(t'), q_i(t')]$, where $p_i(t')$ and $q_i(t')$ are the position and velocity of the particle i at time t', respectively.

Correct answer: State $a = [x_1(t' + M\Delta t), \ldots, x_k(t' + M\Delta t)]$ for some time difference Δt and where $x_i(t)$ is the state of particle i given by its position and velocity at time t.

Network structure: Network depicted in Fig. 9.5 with two evolution steps of the GNN implementing the encoder and two-dimensional edge embeddings $h^L_{i,j} \in \mathbb{R}^2$. The details are given in [184].

Training: As described in Sect. 9.7.2. Further details are given in [184].

Key findings: *SciNet* recovers with high accuracy how the particles interact with each other by classifying their interactions as "no interaction" or "connected by a spring".

Chapter 11
Future Research Directions and Further Reading

In the following, we point out two directions for future research on representation learning for physics. The first direction considers the automatization of searching for strategies to collect relevant observation data. The second points out that it can be challenging to interpret *SciNet*'s representation for data where we do not have a hypothesized representation. We suggest that searching for mathematical expressions for the encoding and decoding mappings may help to tackle this challenge. In addition, such an approach could increase *SciNet*'s ability to generalize to observations that are substantially different than the ones seen during training.

11.1 Finding Measurement Strategies and Representations

As mentioned in the introduction (Chap. 1), deciding how to collect observations given some experimental devices can be challenging. In our examples considered in Chap. 10, observing a time series of a particle or performing random measurements on quantum systems provides sufficient information about the physical systems to response to the asked questions. However, in general, more advanced observation strategies might be required. For example, to determine a position of a particle in a box, one may first use a measurement device that scans the whole box and returns the position of the particle with low accuracy. Then, one may use a second, more precise measurement device that can only scan a small part of the box. This device might target the approximate position of the particle found by the first measurement device to determine the position of the particle with high accuracy. Hence, the optimal setting of the second measurement device depends on the output of the first one. Choosing a random part of the box that is scanned by the second measurement device would mostly lead to not finding the particle and hence, the accuracy for the position measurement would stay low.

To find such advanced measurement strategies, one may train a reinforcement learning agent who gets a reward if he predicts the position of the particle in the box with high accuracy. However, to train such an agent in a supervised way, we must

https://doi.org/10.1007/978-3-031-27019-2_11

know what variables we are interested in (in the example above, it is the position of the particle). In the setting considered in this part of the book, we only know that the observed information should be sufficient to response to the questions that we ask *SciNet*. Hence, following this spirit, the measurement strategy must be learned in parallel with how to respond to the questions with the collected information. A simple example of a question for the example given above would be to ask *SciNet* to predict the electrical field at a fixed point in the box. Assuming the particle is charged, the electrical field might be a function $f(x)$ of the position of the particle. In [240], it is demonstrated on the basis of some toy examples, that current machine learning tools can be adapted to learn the measurement strategy and the function f in parallel. It would be interesting to further investigate this research direction and to see if one can learn observation strategies and representations in parallel for more complex systems. State representation learning, discussed in Remark 11.1, is a direction in machine learning which is closely related to this goal.

Remark 11.1 (State representation learning) In contrast to representation learning, where relevant features are extracted from a (fixed) data set, in state representation learning one considers interactive settings [241]. Thereby, a reinforcement learning agent can interact with an environment and influence the state of the system by its actions. The goal of state representation learning is then to construct a low dimensional representation of the system that evolves in time accordingly to the chosen actions of the agent. In particular, such a setting provides new possible priors for disentangling the representation because the agent can experiment with different actions and observe their effects [242]. For instance, in [52], the representation is disentangled by an *independence prior*, which encourages the storage of independently controllable features of the environment in separate parameters (see also [243] for a similar approach). As demonstrated in [53, 243], such requirements can lead to natural representations in some scenarios such as creating an abstract representation of a labyrinth for navigational tasks. However, in contrast to an operational meaningful representation that depends on the agents' aims (Sect. 8.4.2.2), such representations depend on which actions (and policies) the agent can perform. In physics, these actions depend on the experimental devices and may be considered less fundamental than the agents' aims. Closer to the approach of disentangling a representation with respect to different questions (Sect. 8.4.2.2), in [244] several reinforcement learning agents with different goals share a common representation which is optimzed during training. Interestingly, they find that learning auxiliary tasks can improve the performance of the agent in learning of the overall objective. A further step was taken in [245], where the auxiliary tasks do not have to be specified by humans, but are learned during the training process. In a similar spirit, it would be interesting to learn the relevant questions for *SciNet* instead of choosing them by hand in future work.

11.2 Interpretability and Generalization of *SciNet*

A lot of effort in machine learning is currently invested in improving the interpretability of artificial neural networks (see e.g. [246] for a recent review). For example, inceptionism techniques [247, 248] can be used to gain some insight into what a neural network has learned. Such techniques were used in [249] to extract some relevant properties from thermodynamic systems. Building prior knowledge about the training data into the network structure often improves interpretability. Usually, the interpretability is related to the generalization power of the neural network, and an improvement in interpretability also increases the generalization power. For example, building in information about the object structure of physical systems, as described in Sect. 8.6 and Sect. 9.7, allows the network to generalize to systems with different numbers and configurations of objects and relations. At the same time, the additional structure in the network helps to extract some information about what the network has learned. Indeed, it is demonstrated in Sect. 10.8, how one can extract information about the interaction types between particles from *SciNet*'s representation. On the other hand, building some mathematical prior knowledge into *SciNet* (see Sect. 9.6 for an example) can help to interpret the found representation as demonstrated for the nonlinear pendulum in [183] (and discussed in Sect. 10.3).

For the long term goal of applying the methods discussed in this book for the foundations of physics, one must be very careful building in prior knowledge into *SciNet*'s network structure. In particular, by assuming an object structure of the observation data as considered in Sect. 8.6, it could be that we separate objects in a way that is not "natural" and that there is a more elegant description of the system with a different partitioning of the system into subsystems. Hence, in this part of the book, we focussed on extracting physical concepts from observation data with minimal prior knowledge about physics and mathematics. Such an approach has the advantage that we do not restrict the set of models that can be learned by the neural network. On the other hand, this is a tradeoff with the generalization power of *SciNet* as well as with its interpretability. To interpret the compressed representation found by *SciNet* without using mathematical or physical prior knowledge, we have to plot it against a hypothesized representation. The challenge for future work is hence to find other methods that use a minimal amount of prior knowledge, and still allow to extract some conceptual information from the machine learning system without any known hypothesis.

One possibility to improve the interpretability as well as the generalization power of *SciNet* would for example be to include a bias that is very common in physics: given the relevant variables and parameters of a physical system, we assume that there is a "simple" mathematical expression that predicts the (future) physical properties of the system (at least for small time steps). Hence, more concretely, we may assume that the decoders of *SciNet* can be described by simple mathematical expressions. A step in this direction was recently done in [250]. The authors in [250] use Equation Learner (EQL) networks, similar to the work in [251, 252], which use activation functions that represent simple mathematical operations such as multiplication or taking the sine or cosine. Further, to find “simple” mathematical expressions, they enforce sparsity for the weights in the network. The more weights are set to zero, the shorter the mathematical expression represented by the EQL network becomes. In [250], EQL networks are then applied to different problems, including a dynamical system analysis of a simple kinematic problem and a harmonic oscillator. They use a similar network structure as the one used in Sect. 9.4 to find representations with simple update rules (Fig. 9.3) with a trivial decoder and where the update rules are parametrized by an EQL network. In both examples, the EQL network is able to find simple mathematical expressions for the time evolution of the physical systems. It would be interesting to replace the decoders in *SciNet* with EQL networks and try to extract simple mathematical expressions from them. Such expressions correspond to applications of physical laws in our setting and hence may help to discover the underlying laws themselves. Further, finding mathematical expressions for the encoder could also help to interpret the representation found by *SciNet*. However, for physical examples, the mapping from observation data to the representation is typically more complex than the one from the representation to the predictions. Hence, it could be challenging to find such expressions with EQL networks. Alternatively, one may use feed-forward neural networks for the encoder and try to extract the equation describing its mapping from the trained network using methods from symbolic regression [24] (see Sect. 6.2.1). A step in this direction was considered in [129] (see Remark 6.1 for the method introduced in [129]). Recently, the additional structure of graph neural networks was exploited to extract equations from different parts of the network [205, 253]. The authors of [205] apply their techniques to a cosmology example and discover a new analytic formula which can predict the concentration of dark matter from the mass distribution of nearby cosmic structures. In [253] a graph neural network is trained to simulate the dynamics of our solar system’s Sun, planets, and large moons from thirty years of trajectory data. Then, symbolic regression is used to extract an analytical expression for the force law implicitly learned by the graph neural network, which turns out to be equivalent to Newton’s law of gravitation.

Part IV
Future Prospects

In the last part of this book, we discuss future prospects for the automatization of a physicist's discovery process as well as for using machine learning for the foundation of physics. As it is extremely difficult to make time-related predictions about the future development of AI, we will instead focus on pointing out challenging problems that must be solved to fully automatize the physical discovery process. Furthermore, we point out recent steps towards this long-term goal and particularly promising methods that must be further improved in the future to be applicable to real-world problems.

Chapter 12
Future Prospects

In this book, we considered the automatization of several steps in a physicist's discovery process. However, we are still far away from building an AI scientist that is able to replace human scientists. In fact, developing an AI physicist is probably almost as difficult as developing general artificial intelligence, which, according to most AI researchers, will still be decades away [254, 255]. On the one hand, there are many open questions in automatizing each single step of a physicist's discovery process, and on the other hand, combining all steps is also a highly nontrivial challenge. In the following sections, we discuss recent progress in automatizing several steps of a physicist's discovery process together and future challenges in this direction (Sect. 12.1), how learning procedures instead of simple functions may help to build general artificial intelligence (Sect. 12.2) and last but not least, how AI may help to solve fundamental problems in physics in the future.

12.1 AI Physicist

The ultimate long-term goal would be to build AI physicists, i.e., systems that act and learn autonomously from a physical environment. There is no "exact" definition of an AI physicist and finally, one could also end up building AI systems that are specialized for certain domains and can communicate with each other to share information. This would be similar to human physicists, who are also experts in certain fields, but could not perform all the tasks that other physicists can do. For example, my performance in building up an experiment with devices in a laboratory would probably be close to the performance of an agent trying out random actions. However, at the current time we are so far away from creating an AI physicist who could fully replace *any* human physicist that we may ignore the fact that the goal is not completely specified. For concreteness, one may think of the following goal: building an agent that can interact with the real world and for example finds the law of gravitation. To test if the agent really found the law of gravitation, one could let it predict the movement of bodies that move according to the gravitational force in a completely new situation that the

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2_12

agent has never seen before. Furthermore, the agent should be able to memorize the found insights, and combine them with future insights to unify different theories. In the following, we first describe some recent progress in building AI physicists, and then discuss some essential challenges that are left for future work on building AI physicists.

The first machine in history able to discover new scientific knowledge independently of its human creators was the robot “Adam” [256]. Adam can autonomously perform the scientific cycles of setting up an hypothesis, physically run experiments using a laboratory robot to test the hypothesis and finally, interpret the results. If Adam falsifies the hypothesis, he repeats the cycle. A more recent step towards an artificial intelligence physicist was taken in [4] by also considering several steps of a physicist’s discovery process together. In contrast to most other work, where a machine learning system is used to fit all the given data at once, in [4] the environment is separated into several subdomains using unsupervised learning. The subdomains are chosen in a way, such that a physical law is assigned with each subdomain and performs well in predicting the future evolution of the physical system in this domain. This is a typical strategy of physicists; choosing and isolating a physical system that is then investigated further. As a toy example, we may consider two regions in the xy-plane, where we have a harmonic potential or a homogeneous gravitational field in region one or two, respectively. Hence, the equations of motion for a particle of mass m are given by $\ddot{x} = -k/m(x - x_0)$ and $\ddot{y} = -k/m(y - y_0)$ with spring constant k and equilibrium position (x_0, y_0) in the first region, and by $\ddot{x} = g_x$ and $\ddot{y} = g_y$ for a gravitational field $g = (g_x, g_y)$ in the second region. The machine learning method suggested in [4] is then able to recognize the two domains and uses two neural networks to predict the time evolution of the particle in each region separately. To improve generalization power and interpretability, symbolic expressions are extracted for the two neural networks and added to a *theory hub* [4]. In the example above, the following expressions might be found for the region with a homogeneous gravitational field $\ddot{x} = 2$ and $\ddot{y} = 3\ \text{m/s}^2$, where we assume that $g_x = 2$ and $g_y = 3\ \text{m/s}^2$ for concreteness. The theory is kept in the memory of the AI physicist and is used to investigate new environments. Furthermore, some basic unification of theories is considered in [4], where parametrized *master theories* are found. In the example considered above, this unification algorithm would recognize that the two formulas $\ddot{x} = 2$ and $\ddot{y} = 3\ \text{m/s}^2$ are of the same form and can be considered as instances of the master theory $\ddot{u} = g_u\ \text{m/s}^2$, for any coordinate u and a parameter g_u.

The approach to build an AI physicist described in [4] goes beyond finding a model for one given physical system and also points towards several challenging problems that have to be solved to build a general AI physicist and which we discuss in the following. First, it becomes clear that one of the main difficulties for a physicist is to decompose the environment into subsystems in such a way that a “simple” description of the subsystems is possible. Furthermore, one may want to choose subsystems whose investigation likely leads to the discovery of new physical concepts. In fact, a modern physicist may not want to consider a standard pendulum, but preferably colliding particles with high energy, because such scenarios are less well understood and can possible lead to new physical insights. Thinking about all the information a human obtains through the sensory organs every second, recognizing subsystems and choosing and isolating them for experiments that may lead to new physical insights is an extremely hard task. A first step towards this goal might be to find compact representations of parts of the environment, since this simplifies investigations. Furthermore, finding representations that are not sufficient to predict the future evolution of a subsystem, points towards incomplete understanding of these systems. Steps towards extracting representations directly from video data were done in [34, 192]. For example, in [34] the position of an object is extracted from raw visual data. Such methods may turn out useful for building future AI physicists.

Another difficulty in building a fully fledged AI physicist which is often ignored, but considered in [4], lies in building on existing theories and unifying observed laws from different systems. Fully automatizing the unification of theories and searching for underlying physical principles might be extremely hard and can be equally difficult to achieve as general AI. The ultimate goal of unification is to improve the generalization power of a physicist. Hence, one may estimate the generalization performance of an AI physicist by testing its prediction power in a completely new environment not seen during training, but with the same underlying laws as the training environment. An active direction of research in machine learning called *transfer learning* considers reusing learned concepts in an environment to a new environment (see [257] for a comprehensive survey). Progress in this area may also lead to progress in generalizing physical theories that work well in certain environments to a theory that applies to all considered environments.

12.2 Learning Procedures Instead of Simple Functions

Currently, neural networks are the most important tool for AI. In this book, we mostly used feedforward neural networks for fitting functions. However, a general AI agent cannot be built using feedforward neural networks, since general AI should be able to perform any algorithm that could be run on a Turing machine. For example, consider an AI physicist for the task of calculating the electrical field at a given point in space, given an arbitrary number n of charged particles creating the field. For any fixed number n, a feedforward neural network could be constructed that takes the positions and charges of the n particles as an input and calculates the electrical field at the given point. However, if the number of particles n is a priori unknown, given any feedforward neural network with m input neurons, we can always choose $n > m/4$ particles (note that for each particle we have four inputs, the three-dimensional position and the charge) making the AI physicist fail in predicting the electrical field, since it can not handle this input size. In addition, even if the input size could be handled by the feedforward neural network, it fully ignores the fact that the field created by each particle can be calculated in the same way. Applying the same rule to sequential inputs is naturally handled by recurrent neural networks (as for example used in Sect. 9.4). It was even shown that recurrent neural networks are computationally universal (also called *Turing-complete*) *if* properly wired, i.e., they can be used to simulate any Turing machine and hence any procedure [258]. However, if something is possible in principle, this does not necessarily mean that it will work in practice. In [259], the capacity of recurrent neural networks is enriched to simplify the solution of algorithmic tasks by adding a large addressable memory. This is analog to Turing's enrichment of finite-state machines by an infinite memory tape, and the authors of [259] call this structure a *Neural Turing Machine* (NTM). In contract to a (standard) Turing machine, an NTM is not programmed by a human but does learn programs from data by its own. By ensuring that the outputs depend smoothly on the parameters describing the program, one can use stochastic gradient descent to efficiently train an NTM. Preliminary results show that an NTM can infer simple algorithms such as copying and sorting [259], but it remains as a challenging task for future work to learn programs for more complex tasks from data.

As mentioned above, apart from the additional computational power of NTMs compared to feedforward neural networks, they may learn much simpler descriptions of a procedure. In the example where one wants to calculate the electrical field created by n particles, an NTM would essentially need to learn that it can sequentially consider the particles, calculate the distance to the position of interest, estimate the field according to Coulomb's law and adding up the estimated field strengths on the go. Increasing the number of particles would not make this program any more complex, but would just make it run longer. This is a much more elegant approach than fitting a function on the full input at once, and hence, learning Coulomb's law for each pair of particles separately. To motivate an AI agent to learn simple algorithms instead of complex once, we need some measure of complexity. One option is to use the *Kolmogorov complexity*, which is defined as the minimal description length of a program that produces the desired outputs for given inputs. Note that the minimal description length depends on the available commands that the given Turing machine can execute, and hence, the choice of these commands introduces some bias. Unfortunately, there is no way to efficiently calculate the Kolmogorov complexity for given input-output pairs. Nonetheless, we could motivate an NTM to choose the resulting program that has the smallest description length it can produce. Although this complexity measure is coming from computational theory, it might be the fundamental reason why physicists work and think as they do.[1] Furthermore, searching for programs with short description length also improves the interpretability of NTMs. I would expect that approaches in direction of building trainable Turing machines and searching for simple programs will be essential for the long term goal of building general AI, however, the methods used to implement such approaches may change a lot in the next few decades.

[1] A similar complexity measure was used in [4] to find the simplest formulas describing experimental data.

12.3 AI for Foundations of Physics

Let us finally come back to one of the main motivations behind this book, namely to use AI to help solving fundamental problems in modern physics. The content of this book is focussed on methods that use a minimal amount of prior knowledge about the physical system at hand. For example, extracting representations of quantum systems in Sect. 10.5, we ensure that the agent performing the measurements acts operationally, i.e., does not make use of any prior knowledge about quantum mechanics. Moreover, the network structure, *SciNet*, used to extract the representation is not specifically adapted to quantum systems and is demonstrated to work as well for classical systems such as the damped pendulum (Sect. 10.2). Furthermore, we found that searching for operationally meaningful representations, one finds the representations of the systems commonly used in physical textbooks. This suggests that the required properties of operationally meaningful representations (Sect. 8.4) are also required by human physicists, however, physicists may actually not be aware of this fundamental reason why they are working with parameters like the mass or the charge of a particle.

Therefore, following the spirit of this approach in future work may lead to a better understanding of “hidden” assumptions made by physicists. If, in the far future, one could build an AI agent that discovers the formalism of quantum mechanics by interacting with quantum systems and only assuming a few “natural” requirements on the working of the AI agent (such as searching for operationally meaningful representations), one could stepwise remove the assumed principles and retrain the AI agent. This could lead to the discovery of alternative formulations of quantum mechanics, and hence could ultimately help to overcome the conceptual problems we are currently facing with quantum theory and which may be caused by a “hidden” assumption that we should better give up.

Appendix A
Interpretation of the Number of Latent Variables

Abstract In Sect. 8.4.1 we defined minimal representations and required that they should contain a minimal number parameters. In this appendix [taken from (Iten et al. in Phys Rev Lett 124:010508, 2020 [22])] we use techniques from differential geometry to state a formal relation between the number of parameters in a minimal representation and the degrees of freedom in the considered physical data. This relation gives the minimal number of latent neurons required for *SciNet* (consisting of only one encoder and probably several decoders) to make accurate prediction the expected physical meaning, namely that it corresponds to the degrees of freedom in the physical data that are required to reply to all questions that we may ask *SciNet*.

We describe the given data with triples $(\mathcal{O}, \mathcal{Q}, a^\star)$, where $\mathcal{O}$ and $\mathcal{Q}$ are the sets containing the observation data and the questions respectively, and the function $a^\star : (o, q) \mapsto a$ sends an observation $o \in \mathcal{O}$ and a question $q \in \mathcal{Q}$ to the correct reply $a \in \mathcal{A}$. Note that the case where *SciNet* consists of several decoders can be cast into this setting by stacking all the questions and answers together and by considering the resulting tuples as one question or answer, respectively. The following results only take one encoder into account.

Intuitively, we say that the triple $(\mathcal{O}, \mathcal{Q}, a^\star)$ has dimension at least n if there exist questions in $\mathcal{Q}$ that are able to capture n degrees of freedom from the observation data $\mathcal{O}$. Smoothness of this "oracle" is a natural requirement, in the sense that we expect the dependence of the answers on the input to be robust under small perturbations. The formal definition follows.

Definition A.1 (*Dimension of a data set*) Consider a data set described by the triple $(\mathcal{O}, \mathcal{Q}, a^\star)$, where $a^\star : \mathcal{O} \times \mathcal{Q} \to \mathcal{A}$, and all sets are real, $\mathcal{O} \subseteq \mathbb{R}^r$, $\mathcal{Q} \subseteq \mathbb{R}^s$, $\mathcal{A} \subseteq \mathbb{R}^t$. We say that this triple has dimension at least n if there exists an n-dimensional submanifold $\mathcal{O}_n \subseteq \mathcal{O}$ and questions $q_1, \ldots, q_k \in \mathcal{Q}$ and a function.

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2

$$\begin{aligned} f: \quad \mathcal{O}_n &\to \mathcal{A}^k := \overbrace{A \times A \times \cdots \times A}^{k} \\ o &\mapsto [a^\star(o, q_1), \ldots, a^\star(o, q_k)] \end{aligned}$$

such that $f : \mathcal{O}_n \to f(\mathcal{O}_n)$ is a diffeomorphism.

Proposition A.1 (**Minimal representation for *SciNet***) *A (sufficient) latent representation for data described by a triple*$(\mathcal{O} \subset \mathbb{R}^r, \mathcal{Q} \subset \mathbb{R}^s, a^\star : \mathcal{O} \times \mathcal{Q} \to \mathcal{A} \subset \mathbb{R}^t)$ *of dimension at leastn requires at least n latent variables.*

Proof By assumption, there is an n-dimensional submanifold $\mathcal{O}_n \subset \mathcal{O}$ and k questions $q_1, \ldots, q_k$ such that $f : \mathcal{O}_n \to \mathcal{I}_n := f(\mathcal{O}_n)$ is a diffeomorphism. We prove the statement by contradiction: assume that there exists a (sufficient) representation described by an encoder $E : \mathcal{O} \to \mathcal{R}_m \subset \mathbb{R}^m$ with $m < n$ latent variables. By sufficiency of the representation, there exists a smooth decoder $D : \mathcal{R}_m \times \mathcal{Q} \to \mathcal{A}$ such that $D(E(o), q) = a^\star(o, q)$ for all observations $o \in \mathcal{O}$ and questions $q \in \mathcal{Q}$. We define the smooth map

$$\begin{aligned} \tilde{D} : \mathcal{R}_m &\to \mathcal{A}^k \\ r &\mapsto [D(r, q_1), \ldots, D(r, q_k)], \end{aligned}$$

and denote the pre-image of $\mathcal{I}_n$ by $\tilde{\mathcal{R}}_m := \tilde{D}^{-1}(\mathcal{I}_n)$.

By sufficiency of the representation, the restriction of the map $\tilde{D}$ to $\tilde{\mathcal{R}}_m$ denoted by $\tilde{D}|_{\tilde{\mathcal{R}}_m} : \tilde{\mathcal{R}}_m \to \mathcal{I}_n$ is a smooth and surjective map. However, by Sard's theorem (see for example [260]), the image $\tilde{D}(\tilde{\mathcal{R}}_m)$ is of measure zero in $\mathcal{I}_n$, since the dimension of the domain $\tilde{\mathcal{R}}_m \subset \mathbb{R}^m$ is at most m, which is smaller than the dimension n of the image $\mathcal{I}_n$. This contradicts the surjectivity of $\tilde{D}|_{\tilde{\mathcal{R}}_m}$ and finishes the proof.

We can consider an autoencoder as a special case of *SciNet*, where we ask always the same question and expect the network to reproduce the observation input. Hence, an autoencoder can be described by a triple $(\mathcal{O}, \mathcal{Q} = \{0\}, a^\star : (o, 0) \mapsto o)$. As a corollary of Proposition A.1, we show that in the case of an autoencoder, the required number of latent variables corresponds to the "relevant" number of degrees of freedom that describe the observation input. The relevant degrees of freedom, which are called *(hidden) generative factors* in this context in representation learning (see for example [49]), may be described by the dimension of the domain of a smooth nondegenerate data generating function H, defined as follows.

Definition A.2 We say that a smooth function $H : \mathcal{G} \subset \mathbb{R}^d \to \mathbb{R}^r$ is nondegenerate if there exists an open subset $\mathcal{N}_d \subset \mathcal{G}$ such that the restriction $H|_{\mathcal{N}_d} : \mathcal{N}_d \to H(\mathcal{N}_d)$ of H on $\mathcal{N}_d$ is a diffeomorphism.

One may think of H as sending a small dimensional representation of the data onto a manifold in a high dimensional space of observations.

Corollary A.1 *(***Minimal representation for an autoencoder***) Let $H : \mathcal{G} \subset \mathbb{R}^d \to \mathcal{O} \subset \mathbb{R}^r$ be a smooth, nondegenerate and surjective (data generating) function, and let us assume that $\mathcal{G}$ is bounded. Then the minimal representation for data described by a triple $(\mathcal{O}, \mathcal{Q} = \{0\}, a^\star : (o, 0) \mapsto o)$ contains d latent variables.*

Proof First, we show the existence of a (sufficient) representation with d latent variables. We define the encoder mapping (and hence the representation) by $E : o \mapsto \text{argmin}[H^{-1}(\{o\})] \in \mathcal{G}$, where the minimum takes into account only the first vector entry.[1] We set the decoder equal to the smooth map H. By noting that $D(E(o), 0) = o$ for all $o \in \mathcal{O}$, this shows that d latent variables are sufficient.

Let us now show that there cannot exist a representation with less than d variables. By definition of a nondegenerate function H, there exists an open subset $\mathcal{N}_d \subset \mathcal{G}$ in $\mathbb{R}^d$ such that $H|_{\mathcal{N}_d} : \mathcal{N}_d \to H(\mathcal{N}_d)$ is a diffeomorphism. We define the function $f : o \in H(\mathcal{N}_d) \mapsto a^\star(o, 0) \in \mathcal{I}$, where $\mathcal{I} = H(\mathcal{N}_d)$. Since f is the identity map and hence a diffeomorphism, the data described by the triple $(\mathcal{O}, \mathcal{Q} = \{0\}, a^\star : (o, 0) \mapsto o)$ has dimension at least d. By Proposition A.1, we conclude that at least d latent variables are required.

[1] Note that any element in $H^{-1}(\{o\})$ could be chosen.

Appendix B
Variational Autoencoders

Abstract In this appendix [taken from (Iten et al. in Phys Rev Lett 124:010508, 2020 [22])], we describe variational autoencoders (VAEs) (Higgins et al. in ICLR, 2017 [49], Kingma and Welling in 2013 [193]), a tool from machine learning that can be used to find compact representations of data sets. In particular, we consider beta-VAE (Higgins et al. in ICLR, 2017 [49], an extension of VAEs that is meant to disentangle the parameters in the latent representation. We use beta-VAEs in this thesis as a tool to find statistically independent representations (Sect. 8.4.2.1).

The implementation of *SciNet* uses a modified version of so-called variational autoencoders (VAEs) [49, 193]. The standard VAE architecture does not include the question input used by *SciNet* and tries to reconstruct the input from the representation instead of answering a question. VAEs are one particular architecture used in the field of representation learning [27]. Here, we give a short overview over the goals of representation learning and the details of VAEs.

Representation learning.

The goal in representation learning is to map a high-dimensional input vector x to a lower-dimensional representation $z = (z_1, z_2, \dots, z_d)$, commonly called the *latent vector*.[2] The representation z should still contain all the relevant information about x. In the case of an autoencoder, z is used to reconstruct the input x. This is motivated by the idea that the better the (low-dimensional) representation is, the better the original data can be recovered from it. Specifically, an autoencoder uses a neural network (*encoder*) to map the input x to a small number of latent neurons z. Then, another neural network (*decoder*) is used to reconstruct an estimate of the input, that is $z \mapsto \tilde{x}$. During training, the encoder and decoder are optimized to maximize the reconstruction accuracy and reach $\tilde{x} \approx x$.

[2] The variables x and z correspond to the observation o and the representation r used in the main text.

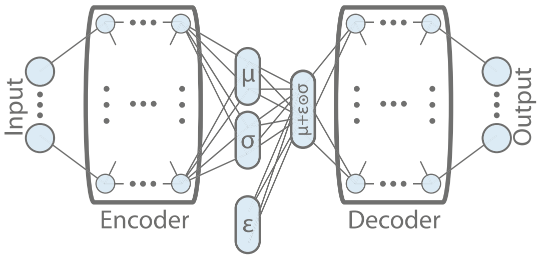


Fig. B.1 Network structure for a variational autoencoder [22]. The encoder and decoder are described by conditional probability distributions $p(z|x)$ and $p(x|z)$ respectively. The output distribution of the encoder are the parameters μ_i and $\log(\sigma_i)$ for independent Gaussian distributions $z_i \sim \mathcal{N}(\mu_i, \sigma_i)$ of the latent variables. The reparameterization trick is used to sample from the latent distribution

Probabilistic encoder and decoder.

Instead of considering deterministic maps $x \mapsto z$ and $z \mapsto \tilde{x}$, we generalize to conditional probability distributions $p(z|x)$ for the encoder and $p(\tilde{x}|z)$ for the decoder. This is motivated by the Bayesian view that the most informative statement the encoder can output a description of a probability distribution over all latent vectors, instead of outputting a single estimate. The same reasoning holds for the decoder. We use the notation $z \sim p(z)$ to indicate that z is picked at random according to the distribution p.

We cannot treat the general case analytically, so we make restricting assumptions to simplify the setting. First we assume that the input can be perfectly compressed and reconstructed by an encoder and decoder which are both neural networks, that is we assume that the ideal distributions $p(z|x)$ and $p(\tilde{x}|z)$ that reach $x = \tilde{x}$ are members of parametric families $\{p_\phi(z|x)\}_\phi$ and $\{p_\theta(\tilde{x}|z)\}_\theta$, respectively. We further assume that it is possible to achieve this with a latent representation where each neuron is independent of the others, $p_\phi(z|x) = \prod_i p_\phi(z_i|x)$. If these distributions turn out hard to find for a given dimension d of the latent representation, we can try to increase the number of neurons of the representation to disentangle them. Finally, we make one more simplifying assumption, which is justified *a posteriori* by good results: that we can reach a good approximation of $p(z|x)$ by using only independent normal distributions for each latent neuron, $p_\phi(z_i|x) = \mathcal{N}(\mu_i, \sigma_i)$, where μ_i is the mean and σ_i the variance. We can think of the encoder as mapping x to the vectors $\mu = (\mu_1, \dots, \mu_d)$ and $\sigma = (\sigma_1, \dots, \sigma_d)$.

The optimal settings for ϕ and θ are then learned as follows, see Fig. B.1:

1. The encoder with parameters (weights and biases) ϕ maps an input x to $p_\phi(z|x) = \mathcal{N}[(\mu_1, \dots, \mu_d), (\sigma_1, \dots, \sigma_d)]$.
2. A latent vector z is sampled from $p_\phi(z|x)$.
3. The decoder with parameters (weights and biases) θ maps the latent vector z to $p_\theta(\tilde{x}|z)$.
4. The parameters ϕ and θ are updated to maximize the likelihood of the original input x under the decoder distribution $p_\theta(\tilde{x}|z)$.

Reparameterization trick.

The operation that samples a latent vector z from $p_\phi(z|x)$ is not differentiable with respect to the parameters ϕ and θ of the network. However, differentiability is necessary to train the network using stochastic gradient descent. This issue is solved by the reparameterization trick introduced in [193]: if $p_\phi(z_i|x)$ is a Gaussian with mean μ_i and standard deviation σ_i, we can replace the sampling operation using an auxiliary random number $\epsilon_i \sim \mathcal{N}(0, 1)$. Then, a sample of the latent variable $z_i \sim \mathcal{N}(\mu_i, \sigma_i)$ can be generated by $z_i = \mu_i + \sigma_i \epsilon_i$. Sampling ϵ_i does not interfere with the gradient descent because ϵ_i is independent of the trainable parameters ϕ and θ. Alternatively, one can view this way of sampling as injecting noise into the latent layer [261].

β-VAE cost function.

A computationally tractable cost function for optimizing the parameters ϕ and θ was derived in [193]. This cost function was extended in [49] to encourage independency of the latent variables $z_1, \ldots, z_d$ (or to encourage "disentangled" representations, in the language of representation learning). The cost function in [49] is known as the β-VAE cost function,

$$C_\beta(x) = -\left[e_{z\sim p_\phi(z|x)} \ \log p_\theta(x|z)\ \right] + \beta\ D_{\mathrm{KL}}\left[p_\phi(z|x)\|h(z)\right],$$

where the distribution $h(z)$ is a prior over the latent variables, typically chosen as the unit Gaussian[3], $\beta \geq 0$ is a constant, and D_{KL} is the Kullback-Leibler (KL) divergence, which is a quasi-distance[4] measure between probability distributions,

$$D_{\mathrm{KL}}\left[p(z)\|q(z)\right] = \sum_z p(z) \log\left(\frac{p(z)}{q(z)}\right).$$

Let us give an intuition for the motivation behind the β-VAE cost function. The first term is a log-likelihood factor, which encourages the network to recover the input data with high accuracy. It asks "for each z , how likely are we to recover the original x after the decoding?" and takes the expectation of the logarithm of this likelihood $p_\theta(x|z)$ (other figures of merit could be used here in an alternative to the logarithm) over z sampled from $p_\phi(z|x)$, in order to simulate the encoding. In practice, this expectation is often estimated with a single sample, which works well enough if the mini-batches are chosen sufficiently large [193].

The second term encourages disentangled representations, and we can motivate it using standard properties of the KL divergence. Our goal is to minimize the amount of correlations between the latent variables z_i: we can do this by minimizing the distance $D_{\mathrm{KL}}\left[p(z)\|\prod_i p(z_i)\right]$ between $p(z)$ and the product of its marginals. For any other distribution with independent z_i, $h(z) = \prod_i h(z_i)$, the KL divergence satisfies

[3] The interpretation of $h(z)$ as a prior is clear only when deriving VAEs as generative networks. For details, see [193].

[4] The KL divergence satisfies all axioms of a metric apart from symmetry.

$$D_{\mathrm{KL}}\left[p(z)\|\prod_i p(z_i)\right] \leq D_{\mathrm{KL}}\left[p(z)\|h(z)\right].$$

The KL divergence is furthermore jointly convex in its arguments, which implies

$$\begin{aligned} D_{\mathrm{KL}}&\left[\sum_x p(x)\ p_\theta(z|x)\|h(z)\right] \\ &\leq \sum_x p(x)\ D_{\mathrm{KL}}\left[p_\theta(z|x)\|h(z)\right]. \end{aligned}$$

Combining this with the previous inequality, we obtain

$$\begin{aligned} D_{\mathrm{KL}}&\left[p(z)\ \|\prod_i p(z_i)\right] \\ &\leq e_{x\sim p(x)}\ D_{\mathrm{KL}}\left[p(z|x)\|h(z)\right]. \end{aligned}$$

The term on the right hand side corresponds exactly to the second term in the cost function, since in the training we try to minimize $e_{x\sim p(x)}C_\beta(x)$. Choosing a large parameter β also penalizes the size of latent representation z, motivating the network to learn an efficient representation. For an empirical test of the effect of large β see [49], and for another theoretical justification using the information bottleneck approach see [261].

To derive an explicit form of C_β for a simple case, we again assume that $p_\phi(z|x) = \mathcal{N}(\mu, \sigma)$. In addition, we assume that the decoder output $p_\theta(\tilde{x}|z)$ is a multivariate Gaussian with mean $\hat{x}$ and fixed covariance matrix $\hat{\sigma} = \frac{1}{\sqrt{2}}\mathrm{id}$. With these assumptions, the β-VAE cost function can be explicitly written as

$$C_\beta(x) = \|\hat{x} - x\|_2^2 - \frac{\beta}{2}\left(\sum_i \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2\right) + C. \qquad \text{(B.1)}$$

The constant terms C do not contribute to the gradients used for training and can therefore be ignored.

Appendix C
Implementation Details

Abstract In this appendix [which is based on (Iten et al. in Phys Rev Lett 124:010508, 2020 [22])] we provide some details about the implementations used for the examples discussed in Chap. 10. We specify the sizes of the neural networks that were used and provide the most relevant parameters used to train the networks. The full details can be found in the open source code.

The following implementation details refer to Sects. 10.2, 10.4, 10.5, 10.6 and 10.7. For the implementation details for the examples given in Sects. 10.3 and 10.8, we refer to [183, 184], respectively, and their open source code given here:

https://github.com/BethanyL/DeepKoopman and https://github.com/ethanfetaya/nri.

The neural networks used in this work are specified by the input sizes for question and observation inputs, the output sizes, the number of latent neurons, and the sizes of encoders and decoders. The number of neurons in the encoders and decoders are not expected to be important for the results, provided the encoders and decoders are large enough to not constrain the expressivity of the network (and small enough to be efficiently trainable). The training is specified by the number of training examples, the batch size, the number of epochs, the learning rate and the value of the parameter β, which regulates the tradeoff between disentangling and accurate predictions. Note that dependent on the considered example, β corresponds to the parameter in the cost function of the beta-VAE given in (B.1) or the one appearing in the cost function to find operationally meaningful representations given in (9.1), respectively. To test the networks, we use a number of previously unseen test samples. We give the values of these parameters for the examples presented in Chap. 10 in Tables C.1 and C.2.

The source code, all details about the network structure and training process, and pre-trained *SciNets* are available at

https://github.com/eth-nn-physics/nn_physical_concepts and https://github.com/tonymetger/communicating_scinet.

The networks were implemented using the Tensorflow library [212]. For all examples, the training process only takes a few hours on a standard laptop.

Table C.1 Parameters specifying the network structure for the examples in Chap. 10

Example	Observation input size	Question input size	# Latent neurons	Output size	Encoder	Decoder
Pendulum	50	1	3	1	[500, 100]	[100, 100]
Collision	30	16	1	2	[150, 100]	[100, 150]
One qubit (pure state)	10	10	0–5	1	[100, 100]	[100, 100]
Two qubits (pure state)	30	30	0–9	1	[300, 100]	[100, 100]
Two qubits (mixed state)	75	[75, 75, 75]	20	[1, 1, 1]	[500, 250]	[[500, 100], [500, 100], [500, 100]]
Charged particles (single encoder)	40	[1, 1, 1, 1]	3	[1,1,1,1]	[500, 250]	[[100, 100], [100, 100], [100, 100], [100, 100]]
Charged particles (multiple encoder)	[20, 20]	[1, 1, 1, 1]	[2, 2]	[1, 1, 1, 1]	[[200, 200], [200, 200]]	[[100, 100], [100, 100], [100, 100], [100, 100]]
Solar system	2	0	2	2	[100, 100]	[100, 100]

The notation $[n_1, n_2]$ is used to describe the number of neurons n_1 and n_2 in the first and the second hidden layer of the encoders (or the decoders), respectively. If several instances of an object are used, they are listed in a list, e.g. the entry [[200, 200], [200, 200]] in the column "Encoder" means that two encoders were used, where each of them consists of two hidden layers consisting of 200 neurons

Table C.2 Parameters specifying the training process for the examples in Chap. 10

Example	Batch size	Learning rate	β	# Epochs	# Training samples	# Test samples
Pendulum	512	10^{-3}	10^{-3}	1000	95,000	5000
Collision	500	$(5 \cdot 10^{-4}, 10^{-4})$	0	(100, 50)	490,000	10,000
One qubit (pure state)	512	$(10^{-3}, 10^{-4})$	10^{-4}	(250, 50)	95,000	5000
Two qubits (pure state)	512	$(10^{-3}, 10^{-4})$	10^{-4}	(250, 50)	490,000	10,000
Two qubits (mixed state)	512	$5 \cdot 10^{-4}$	$2 \cdot 10^{-5}$	1000	95,000	5000
Charged particles (single encoder)	512	10^{-3}	10^{-3}	5000	95,000	5000
Charged particles (multiple encoder)	256	10^{-3}	10^{-3}	5000	95,000	5000
Solar system	256–1024	$(0.5–3) \cdot 10^{-3}$	$(0.5–1) \cdot 10^{-2}$	5000	95,000	5000

For training with two phases, the notation (p_1, p_2) refers to the parameters in the first and second phase, respectively. The last example uses five training phases specified in detail in [262]. Some less relevant parameters used for regularization during the training are not listed here, but can be found in [263, 264]

Appendix D
Representations of Cyclic Parameters

Abstract In this appendix [which is taken from (Iten et al. in Phys Rev Lett 124:010508, 2020 [22])] We explain the difficulty of a neural network to learn representations of cyclic parameters, which was alluded to in the context of the qubit example [Sect. 10.5, see (Pitelis et al. in IEEE Conference on Computer Vision and Pattern Recognition, 2013 [265], Korman in 2018 [266]) for a detailed discussion relevant to computer vision]. In general, this problem occurs if the data $\mathcal{O}$ that we would like to represent forms a closed manifold (i.e., a compact manifold without boundary), such as a circle, a sphere or a Klein bottle. In that case, several coordinate charts are required to describe this manifold.

Let us start with an example: We consider data points lying on the unit sphere $\mathcal{O} = \{(x, y, z) : x^2 + y^2 + z^2 = 1\}$, which we would like to encode into a simple representation. The data can be (globally) parameterized with spherical coordinates $\phi \in [0, 2\pi)$ and $\theta \in [0, \pi]$ where $(x, y, z) = f(\theta, \phi) := (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$.[5] We would like the encoder to perform the mapping f^{-1}, where we define $f^{-1}((0, 0, 1)) = (0, 0)$ and $f^{-1}((0, 0, -1)) = (\pi, 0)$ for convenience. This mapping is not continuous at points on the sphere with $\phi = 0$ for $\theta \in (0, \pi)$. Therefore, using a neural network as an encoder leads to problems, as neural networks, as introduced here, can only implement continuous functions. In practice, the network is forced to approximate the discontinuity in the encoder by a very steep continuous function, which leads to a high error for points close to the discontinuity.

In the qubit example, the same problem appears. To parameterize a qubit state ψ with two parameters, the Bloch sphere with parameters $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$ is used: the state ψ can be written as $\psi(\theta, \phi) = (\cos(\theta/2), e^{i\phi}\sin(\theta/2))$ (see for example [72] for more details). Ideally, the encoder would perform the map $E : o(\psi(\theta, \phi)) := \left(|\langle\alpha_1|\psi(\theta, \phi)\rangle|^2, \ldots, |\left\langle\alpha_{N_1}|\psi(\theta, \phi)\right\rangle|^2\right) \mapsto (\theta, \phi)$ for some fixed binary projective measurements $\alpha_i \in \mathbb{C}^2$. However, such an encoder is not con-

[5] The function f is not a chart, since it is not injective and its domain is not open.

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2

tinuous. Indeed, assuming that the encoder is continuous, leads to the following contradiction:

$$\begin{aligned}(\theta, 0) &= E(o(\psi(\theta, \phi = 0))) \\ &= E(o(\lim_{\phi \to 2\pi} \psi(\theta, \phi))) \\ &= \lim_{\phi \to 2\pi} E(o(\psi(\theta, \phi))) \\ &= \lim_{\phi \to 2\pi} (\theta, \phi) = (\theta, 2\pi)\,,\end{aligned}$$

where we have used the periodicity of ϕ in the second equality and the fact that the Bloch sphere representation and the scalar product (and hence $o(\psi(\theta, \phi))$) as well as the encoder (by assumption) are continuous in ϕ in the third equality.

Appendix E
Classical Mechanics Derivation for Charged Masse

Abstract In Sect. 10.6 we consider a physical setting with two charged particles, where an agent chooses the angle of a projectile that hits one of the particles. The aim of the agent is to choose the angle such that the particle lands in a hole, which is at a fixed position. In this appendix [which is taken from (Nautrup et al. in Mach Learn Sci Technol 3(4):045025, 2022 [23], Nautrup et al. in Mach Learn Sci Technol, 3:045025 2022 [237])], we derive the analytical solution for this setting, which is then used for supervised training of *SciNet*'s predictions.

In this section, we provide the analytic solution to the charged masses example in Sect. 10.6 that we use to evaluate the cost function for training the neural networks. This is a fairly direct application of the generic Kepler problem, but we include the derivation for the sake of completeness. We use the notation of [267].

The setup we consider is shown in Fig. E.1. Our goal is to derive a function $v_0(\phi)$ that, for fixed q, Q, d_0 and given ϕ, outputs an initial velocity for the left mass such

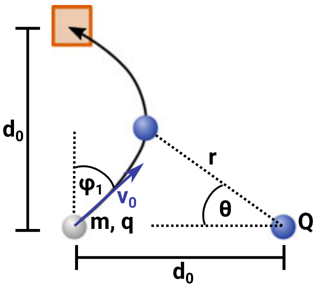


Fig. E.1 Setup and variable names for a charged mass being shot into a holes [23, 237]. A charged particle with mass m and charge q moves in the electrostatic field generated by another charge Q at a fixed position. The initial conditions are given by the velocity v_0 and the angle ϕ. We want to determine the value for ϕ that will result in the particle landing in the target hole, given a velocity v_0

R. Iten, *Artificial Intelligence for Scientific Discoveries*,
https://doi.org/10.1007/978-3-031-27019-2

that the mass will reach the hole. Introducing the inverse radial coordinate $u = \frac{1}{r}$, the orbit $r(\theta)$ of the left mass obeys the following differential equation (see for example [267, Sect. 4.3]):

$$\frac{d^2u}{d\theta^2} + u = \frac{k}{l^2}\,, \tag{E.1}$$

with the constant

$$k = \frac{-qQ}{4\pi\epsilon_0 m} \tag{E.2}$$

and the mass-normalised angular momentum

$$l = r^2\frac{d\theta}{dt}\,. \tag{E.3}$$

This is a conserved quantity and we can determine it from the initial condition of the problem

$$l = d_0 v_0 \cos\phi\,. \tag{E.4}$$

The general solution to Eq. (E.1) is given by

$$u = A\cos(\theta - \theta_0) + \frac{k}{l^2}\,, \tag{E.5}$$

where A and θ_0 are constants to be determined from the initial conditions. The initial conditions are

$$r(\theta = 0) = \frac{1}{A\cos(\theta_0) + \frac{k}{l^2}} = d_0\,, \tag{E.6}$$

$$\left.\frac{dr}{d\theta}\right|_{\theta=0} = \frac{-A\sin\theta_0}{\left(A\cos\theta_0 + \frac{k}{l^2}\right)^2}\frac{v_0\cos\phi}{d_0} = v_0\sin\phi\,. \tag{E.7}$$

Combining these yields

$$A\cos\theta_0 = \frac{1}{d_0} - \frac{k}{l^2}\,, \tag{E.8}$$

$$A\sin\theta_0 = -\frac{1}{d_0}\tan\phi\,. \tag{E.9}$$

The condition that the mass reaches the hole is expressed in terms of $r(\theta)$ as follows:

$$r\left(\theta = \frac{\pi}{4}\right) = \frac{1}{A\cos(\frac{\pi}{4} - \theta_0) + \frac{k}{l^2}} = \sqrt{2}d_0\,. \tag{E.10}$$

Using $\cos(\pi/4 - \theta_0) = \cos(\theta_0)/\sqrt{2} + \sin(\theta_0)/\sqrt{2}$ and the definition of l as well as Eqs. (E.8) and (E.9), we can solve this for v_0:

$$v_0^2 = \frac{(\sqrt{2}-1)k}{d_0} \frac{1}{\cos\phi \sin\phi} \,. \tag{E.11}$$

Restricting ϕ to a suitably small interval, this function is injective and has a well-defined inverse $\phi(v_0)$. The neural network has to compute this inverse from operational input data. To generate valid question-answer pairs, we evaluate $v_0(\phi)$ on a large number of randomly chosen ϕ (inside the interval where the function is injective).

References

1. Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., & Silver, D. (2020). *Nature, 588*(7839), 604.
2. Segler, M. H. S., Preuss, M., & Waller, M. P. (2018). *Nature, 555*(7698), 604.
3. Dalgaard, M., Motzoi, F., Sørensen, J. J., & Sherson, J. (2020). *NPJ Quantum Information*, *6*(1), 6. https://doi.org/10.1038/s41534-019-0241-0
4. Wu, T., & Tegmark, M. (2018). http://arxiv.org/abs/1810.10525
5. Boyle, R. (1772). *The works of the honourable Robert Boyle* (2nd ed, Vol. 6).
6. Wigner, E. P. (1995). The collected works of Eugene Paul Wigner. In J. Mehra (Ed.), *Philosophical reflections and syntheses* (pp. 247–260). Springer. https://doi.org/10.1007/978-3-642-78374-6_20
7. Deutsch, D., & Penrose, R. (1985). Proceedings of the Royal Society of London. *A. Mathematical and Physical Sciences, 400*(1818), 97.
8. Brukner, Č. (2017). The frontiers collection. In R. Bertlmann & A. Zeilinger (Eds.), *Quantum [Un]speakables II: Half a Century of Bell's Theorem* (pp. 95–117). Springer International Publishing. https://doi.org/10.1007/978-3-319-38987-5_5
9. Frauchiger, D., & Renner, R. (2018). *Nature Communications, 9*(1), 3711.
10. Proietti, M., Pickston, A., Graffitti, F., Barrow, P., Kundys, D., Branciard, C., Ringbauer, M., & Fedrizzi, A. (2019). *Science Advances*, *5*(9). https://advances.sciencemag.org/content/5/9/eaaw9832
11. Bong, K. W., Utreras-Alarcón, A., Ghafari, F., Liang, Y. C., Tischler, N., Cavalcanti, E. G., Pryde, G. J., & Wiseman, H. M. (2020). *Nature Physics*. https://www.nature.com/articles/s41567-020-0990-x. Publisher: Nature Publishing Group.
12. Hawking, S. W. (1976). *Physics Review D, 14*, 2460.
13. Preskill, J. (1993). *International Symposium on Black Holes*. Membranes: Wormholes, and Superstrings.
14. Dunjko, V., & Briegel, H. J. (2017). http://arxiv.org/abs/1709.02779
15. Roscher, R., Bohn, B., Duarte, M. F., & Garcke, J. (2019). http://arxiv.org/abs/1905.08883
16. Alhousseini, I., Chemissany, W., Kleit, F., & Nasrallah, A. (2019). http://arxiv.org/abs/1905.01023
17. Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., & Zdeborová, L. (2019). http://arxiv.org/abs/1903.10563
18. Crutchfield, J. P., & McNamara, B. S. (1987). *Complex Systems, 1*(3), 417.

19. Schmidt, M., & Lipson, H. (2009). *Science, 324*, 81. https://doi.org/10.1126/science.1165893
20. Schmidt, M., Vallabhajosyula, R. R., Jenkins, J. W., Hood, J. E., Soni, A. S., Wikswo, J. P., & Lipson, H. (2011). *Physical Biology, 8*(5), 055011. https://doi.org/10.1088/1478-3975/8/5/055011
21. Koza, J. R. (1994). *Statistics and Computing, 4*(2), 87.
22. Iten, R., Metger, T., Wilming, H., del Rio, L., & Renner, R. (2020). *Physics Review Letter 124*, 010508. https://doi.org/10.1103/PhysRevLett.124.010508
23. Nautrup, H. P., Metger, T., Iten, R., Jerbi, S., Trenkwalder, L. M., Wilming, H., Briegel, H. J., & Renner, R. (2022). *Machine Learning: Science and Technology, 3*(4), 045025.
24. Koza, J. R. (1994). *Statistics and Computing, 4*(2), 87.
25. Forrest, S. (1993). *Science, 261*(5123), 872.
26. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25*, (pp. 1097–1105). Curran Associates, Inc.
27. Bengio, Y., Courville, A., & Vincent, P. (2012). *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*. https://arxiv.org/abs/1206.5538?context=cs
28. Geman, S., Bienenstock, E., & Doursat, R. (1992). *Neural Computation*. https://doi.org/10.1162/neco.1992.4.1.1
29. Bates, C., Yildirim, I., Tenenbaum, J. B., & Battaglia, P. W. (2015). *Proceedings of the 37th Annual Conference of the Cognitive Science Society* (pp. 172–77). Pasadena, CA.
30. Wu, J., Yildirim, I., Lim, J. J., Freeman, B., & Tenenbaum, J. (2015). In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28* (pp. 127–135). Curran Associates, Inc.
31. Bramley, N. R., Gerstenberg, T., Tenenbaum, J. B., & Gureckis, T. M. (2018). *Cognitive Psychology, 105*, 9.
32. Rempe, D., Sridhar, S., Wang, H., & Guibas, L. J. (2019). http://arxiv.org/abs/1901.00466
33. Kissner, M., & Mayer, H. (2019). http://arxiv.org/abs/1905.09891
34. Ehrhardt, S., Monszpart, A., Mitra, N. & Vedaldi, A. (2018). http://arxiv.org/abs/1805.05086
35. Ye, T., Wang, X., Davidson, J., & Gupta, A. (2018). http://arxiv.org/abs/1808.10002
36. Lerer, A., Gross, S., & Fergus, R. (2016). http://arxiv.org/abs/1603.01312
37. Nielsen, M. A. (2018). *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/
38. Tishby, N., & Zaslavsky, N. (2015). *IEEE Information Theory Workshop (ITW)* 1–5. https://doi.org/10.1109/ITW.2015.7133169
39. Nair, V., & Hinton, G. (2010). *Proceedings of the 27th International Conference on Machine Learning, Haifa* (pp. 807–814).
40. Glorot, X., Bordes, A. & Bengio, Y. (2010). *International Conference on Artificial Intelligence and Statistics, 15*.
41. Clevert, D. A., Unterthiner, T. & Hochreiter, S. (2015). http://arxiv.org/abs/1511.07289
42. Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017) In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf
43. Cybenko, G. (1989). Mathematics of control. *Signals and Systems, 2*(4), 303.
44. Hornik, K., Stinchcombe, M., & White, H. (1989). *Neural Networks, 2*(5), 359.
45. Sonoda, S., & Murata, N. (2017). *Applied and Computational Harmonic Analysis 43*(2), 233. https://doi.org/10.1016/j.acha.2015.12.005. http://arxiv.org/abs/1505.03654. ArXiv: 1505.03654
46. Pascanu, R., Montufar, G., & Bengio, Y. (2014). arXiv:1312.6098. http://arxiv.org/abs/1312.6098
47. Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09* (pp. 1–8). Montreal, Quebec, Canada: ACM Press. https://doi.org/10.1145/1553374.1553453. http://portal.acm.org/citation.cfm?doid=1553374.1553453

48. Zeiler, M .D., & Fergus, R. (2014). Lecture notes in computer science. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision—ECCV 2014* (pp. 818–833). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-10590-1_53
49. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2017). *ICLR*. https://openreview.net/references/pdf?id=Sy2fzU9gl
50. Chen, T. Q., Li, X., Grosse, R. B., & Duvenaud, D. (2018). http://arxiv.org/abs/1802.04942
51. Kim, H. & Mnih, A. (2018). https://arxiv.org/abs/1802.05983
52. Thomas, V., Bengio, E., Fedus, W., Pondard, J., Beaudoin, P., Larochelle, H., Pineau, J., Precup, D., & Bengio, Y., (2018). http://arxiv.org/abs/1802.09484
53. François-Lavet, V., Bengio, Y., Precup, D., & Pineau, J. (2019). *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019* (pp. 3582–3589). https://doi.org/10.1609/aaai.v33i01.33013582
54. Bengio, Y. (2017). http://arxiv.org/abs/1709.08568
55. Krenn, M., Malik, M., Fickler, R., Lapkiewicz, R., & Zeilinger, A. (2016). *Physics Review Letter, 116*, 090405.
56. Melnikov, A. A., Nautrup, H. P., Krenn, M., Dunjko, V., Tiersch, M., Zeilinger, A., & Briegel, H. J. (2018). *Proceedings of the National Academy of Sciences, 115*(6), 1221.
57. Krenn, M., Erhard, M., & Zeilinger, A. (2020). *Nature Reviews Physics*, *2*(11), 649–661. https://doi.org/10.1038/s42254-020-0230-4
58. Gao, X., Krenn, M., Kysela, J., & Zeilinger, A. (2019). *Physical Review A, 99*(2), 023825. https://doi.org/10.1103/PhysRevA.99.023825
59. Malik, M., Erhard, M., Huber, M., Krenn, M., Fickler, R., & Zeilinger, A. (2016). *Nature Photonics, 10*(4), 248. https://doi.org/10.1038/nphoton.2016.12
60. Schlederer, F., Krenn, M., Fickler, R., Malik, M., & Zeilinger, A. (2016). *New Journal of Physics, 18*(4), 043019. https://doi.org/10.1088/1367-2630/18/4/043019
61. Wang, F., Erhard, M., Babazadeh, A., Malik, M., Krenn, M., & Zeilinger, A. (2017). *Optica, 4*(12), 1462. https://doi.org/10.1364/OPTICA.4.001462
62. Babazadeh, A., Erhard, M., Wang, F., Malik, M., Nouroozi, R., Krenn, M., & Zeilinger, A. (2017). *Physical Reviews, 119*(18), 180510. https://doi.org/10.1103/PhysRevLett.119.180510
63. Erhard, M., Malik, M., Krenn, M., & Zeilinger, A. (2018). *Nature Photonics, 12*(12), 759. https://doi.org/10.1038/s41566-018-0257-6
64. Kysela, J., Erhard, M., Hochrainer, A., Krenn, M., & Zeilinger, A. (2020). *Proceedings of the National Academy of Sciences, 117*(42), 26118.
65. Krenn, M., Hochrainer, A., Lahiri, M., & Zeilinger, A. (2017). *Physical Reviews, 118*(8), 080401. https://doi.org/10.1103/PhysRevLett.118.080401
66. Krenn, M., Gu, X., & Zeilinger, A. (2017). *Physics Review Letter, 119*, 240403.
67. Gao, X., Erhard, M., Zeilinger, A., & Krenn, M. (2020). *Physical Reviews, 125*(5), 050501. https://doi.org/10.1103/PhysRevLett.125.050501
68. Krenn, M., Kottmann, J., Tischler, N., & Aspuru-Guzik, A. (2020). arXiv:2005.06443 [physics, physics:quant-ph] http://arxiv.org/abs/2005.06443. ArXiv: 2005.06443
69. Friederich, P., Krenn, M., Tamblyn, I. & Aspuru-Guzik, A. (2020). Scientific intuition inspired by machine learning generated hypotheses.
70. Briegel, H. J., & Cuevas, G. D. l. (2012). *Scientific Reports, 2*, 400. https://doi.org/10.1038/srep00400, https://www.nature.com/articles/srep00400
71. Bell, J. S. (1964). *Physics Physique Fizika, 1*, 195.
72. Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press. https://doi.org/10.1017/CBO9780511976667
73. Sachdev, S. (2000). *Quantum Phase Transitions*. Cambridge University Press. https://doi.org/10.1017/CBO9780511622540
74. Huber, M., & de Vicente, J. I. (2013). *Physical Reviews 110*(3), 030501. https://doi.org/10.1103/PhysRevLett.110.030501. http://arxiv.org/abs/1210.6876. ArXiv: 1210.6876

75. Horodecki, R., Horodecki, P., Horodecki, M., & Horodecki, K. (2009). *Review Model Physics, 81*, 865.
76. Amico, L., Fazio, R., Osterloh, A., & Vedral, V. (2008). *Review Model Physics 80*, 517.
77. Pan, J. W., Chen, Z. B., Lu, C. Y., Weinfurter, H., Zeilinger, A., & Żukowski, M. (2012). *Review Model Physics, 84*, 777.
78. Hein, M., Eisert, J., & Briegel, H. J. (2004). *Physics Review A, 69*, 062311.
79. Scarani, V., & Gisin, N. (2001). *Physics Review Letter, 87*, 117901.
80. Scarani, V., Bechmann-Pasquinucci, H., Cerf, N. J., Dušek, M., Lütkenhaus, N., & Peev, M. (2009). *Review Model Physics, 81*, 1301.
81. Mautner, J., Makmal, A., Manzano, D., Tiersch, M., & Briegel, H. J. (2015). *New Generation Computing, 33*(1), 69.
82. Melnikov, A. A., Makmal, A., & Briegel, H. J. (2014). arXiv:1405.5459 [cs]. http://arxiv.org/abs/1405.5459. ArXiv: 1405.5459
83. Melnikov, A. A., Makmal, A., Dunjko, V., & Briegel, H. J. (2017). *Scientific Reports, 7*(1), 14430.
84. Melnikov, A. A., Makmal, A., & Briegel, H. J. (2018). *IEEE Access, 6*, 64639. https://doi.org/10.1109/ACCESS.2018.2876494
85. Hangl, S., Ugur, E., Szedmak, S., & Piater, J. (2016). *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2799–2804). https://doi.org/10.1109/IROS.2016.7759434
86. Hangl, S., Dunjko, V., Briegel, H. J., & Piater, J. (2020). *Frontiers in Robotics and AI*, *7*, 42. https://doi.org/10.3389/frobt.2020.00042, https://www.frontiersin.org/article/10.3389/frobt.2020.00042
87. Nautrup, H. P., Delfosse, N., Dunjko, V., Briegel, H. J. & Friis, N. (2019). *Quantum*, *3*, 215. https://doi.org/10.22331/q-2019-12-16-215
88. Tiersch, M., Ganahl, E. J., & Briegel, H. J. (2015). *Scientific Reports, 5*(1), 12874.
89. Wallnöfer, J., Melnikov, A. A., Dür, W., & Briegel, H. J. (2020). *PRX Quantum, 1*, 010301.
90. Melnikov, A. A., Sekatski, P., & Sangouard, N. (2020). *Physics Review Letter 125*, 160401.
91. Ried, K., Müller, T., & Briegel, H. J. (2019). *PLOS ONE, 14*(2), e0212044.
92. López-Incera, A., Ried, K., Müller, T., & Briegel, H. (2020). *PloS one, 15*(12), e0243628.
93. López-Incera, A., Nouvian, M., Ried, K., Müller, T., & Briegel, H. J. (2020). Collective defense of honeybee colonies: Experimental results and theoretical modeling.
94. Paparo, G. D., Dunjko, V., Makmal, A., Martin-Delgado, M. A., & Briegel, H. J. (2014). *Physics Review X, 4*, 031002.
95. Dunjko, V., Friis, N., & Briegel, H. J. (2015). *New Journal of Physics, 17*(2), 023006.
96. Friis, N., Melnikov, A. A., Kirchmair, G., & Briegel, H. J. (2015). *Scientific Reports, 5*(1), 18036.
97. Dunjko, V., Taylor, J. M., & Briegel, H. J. (2016). *Physics Review Letter 117*, 130501. https://link.aps.org/doi/10.1103/PhysRevLett.117.130501
98. Dunjko, V., Taylor, J. M., & Briegel, H. J. (2017). *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 282–287). https://doi.org/10.1109/SMC.2017.8122616
99. Sriarunothai, T., Wolk, S., Giri, G. S., Friis, N., Dunjko, V., Briegel, H. J., & Wunderlich, C. (2018). *Quantum Science and Technology, 4*(1), 015014.
100. Jerbi, S., Trenkwalder, L. M., Poulsen Nautrup, H., Briegel, H. J., & Dunjko, V. (2021). *PRX Quantum*, *2*, 010328. https://doi.org/10.1103/PRXQuantum.2.010328, https://link.aps.org/doi/10.1103/PRXQuantum.2.010328
101. Briegel, H. J., & Müller, T. (2015). *Minds and Machines, 25*(3), 261.
102. Briegel, H. J. (2012). *Scientific Reports, 2*(1), 522.
103. Müller, T., & Briegel, H. J. (2018). *Dialectica, 72*(2), 219.
104. Tulving, E. (1972). *Organization of Memory* (pp. xiii, 423–xiii, 423). Academic Press.
105. Ingvar, D. H. (1985). *Human Neurobiology, 4*(3), 127.
106. Makmal, A., Melnikov, A. A., Dunjko, V., & Briegel, H. J. (2016). *IEEE Access, 4*, 2110. https://doi.org/10.1109/ACCESS.2016.2556579

107. Klyshko, D. N. (1988). *Physics-Uspekhi 31*(1), 74. https://ufn.ru/en/articles/1988/1/f/
108. Leach, J., Padgett, M. J., Barnett, S. M., Franke-Arnold, S., & Courtial, J. (2002). *Physics Review Letter, 88*, 257901.
109. Schütt, K. T., Chmiela, S., von Lilienfeld, O. A., Tkatchenko, A., Tsuda, K. & Müller, K. R. *Machine Learning Meets Quantum Physics*. https://link.springer.com/book/10.1007/978-3-030-40245-7#toc
110. Decelle, A., Martin-Mayor, V., & Seoane, B. (2019). http://arxiv.org/abs/1904.07637
111. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). arXiv:1704.01212, http://arxiv.org/abs/1704.01212
112. De Cao, N., & Kipf, T. (2018). arXiv:1805.11973, http://arxiv.org/abs/1805.11973
113. Krenn, M., Häse, F., Nigam, A., Friederich, P., & Aspuru-Guzik, A. (2020). *Machine Learning: Science and Technology, 1*(4), 045024.
114. Sanchez-Lengeling, B., & Aspuru-Guzik, A. (2018). *Science, 361*(6400), 360. https://doi.org/10.1126/science.aat2663. https://science.sciencemag.org/content/361/6400/360. Publisher: American Association for the Advancement of Science Section: Review.
115. Gromski, P. S., Henson, A. B., Granda, J. M., & Cronin, L. (2019). *Nature Reviews Chemistry, 3*(2), 119. https://doi.org/10.1038/s41570-018-0066-y. https://www.nature.com/articles/s41570-018-0066-y. Number: 2 Publisher: Nature Publishing Group.
116. Bongard, J., & Lipson, H. (2007). *Proceedings of the National Academy of Sciences, 104*(24), 9943. https://doi.org/10.1073/pnas.0609476104, https://www.pnas.org/content/104/24/9943
117. Daniels, B. C., & Nemenman, I. (2015). *Nature Communications, 6*, 8133. https://doi.org/10.1038/ncomms9133
118. Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). *Proceedings of the National Academy of Sciences, 113*(15), 3932.
119. Reinbold, P. A. K., Kageorge, L. M., Schatz, M. F., & Grigoriev, R. O. (2021). *Nature Communications, 12*(1), 3219.
120. Lu, P. Y., Ariño, J. B., & Soljačić, M. (2022). *Communications Physics, 5*(1), 1.
121. Cubitt, T. S., Eisert, J., & Wolf, M. M. (2012). *Physical Reviews, 108*(12), 120503. https://doi.org/10.1103/PhysRevLett.108.120503, https://link.aps.org/doi/10.1103/PhysRevLett.108.120503
122. Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). *Journal of Computational Physics 378*, 686. https://doi.org/10.1016/j.jcp.2018.10.045, http://www.sciencedirect.com/science/article/pii/S0021999118307125
123. Raissi, M. (2018). http://arxiv.org/abs/1801.06637
124. Raissi, M., & Karniadakis, G. E. (2018). *Journal of Computational Physics, 357*, 125. https://doi.org/10.1016/j.jcp.2017.11.039. http://www.sciencedirect.com/science/article/pii/S0021999117309014
125. Zhang, D., Guo, L., & Karniadakis, G. E. (2019). http://arxiv.org/abs/1905.01205
126. Goeßmann, A., Götte, M., Roth, I., Sweke, R., Kutyniok, G., & Eisert, J. (2020). Tensor network approaches for learning non-linear dynamical laws.
127. Forrest, S. (1993). *Science 261*(5123). https://doi.org/10.1126/science.8346439, https://science.sciencemag.org/content/261/5123/872
128. Udrescu, S. M., & Tegmark, M. (2020). *Science Advances*, *6*(16). https://doi.org/10.1126/sciadv.aay2631, https://advances.sciencemag.org/content/6/16/eaay2631
129. Neuweiler, P. (2019). Semester thesis, ETH.
130. Anderson, P. W. (1972). *Science, 177*(4047), 393.
131. Noether, E., König Gesellsch, N. D., & Göttingen, D. W. Z. (1918). *Math-Physics Klasse, 235*
132. Hanc, J., Tuleja, S., & Hancova, M. (2004). *American Journal of Physics, 72*(4), 428. https://doi.org/10.1119/1.1591764
133. Guimerà, R., Reichardt, I., Aguilar-Mogas, A., Massucci, F.A., Miranda, M., Pallarès, J., & Sales-Pardo, M., *Science Advances, 6*(5), 6971. https://doi.org/10.1126/sciadv.aav6971, https://www.science.org/doi/abs/10.1126/sciadv.aav6971

134. Patrignani, C. (2016). *Chinese Physics C*, *40*(10), 100001. https://doi.org/10.1088/1674-1137/40/10/100001. https://iopscience.iop.org/article/10.1088/1674-1137/40/10/100001/meta. Publisher: IOP Publishing.
135. Aaltonen, T., et al. (2008). *Physics Review D, 78*, 012002.
136. Aaltonen, T., et al. (2009). *Physical Review D, 79*(1), 011101.
137. Meyer, A. (2010). *AIP Conference Proceedings, 1200*, 293.
138. Choudalakis, G. (2011). *PHYSTAT 2011, Workshop on Statistical Issues Related to Discovery Claims in Search Experiments and Unfolding*. CERN.
139. Aaboud, M., Aad, G., Abbott, B., Abdinov, O., Abeloos, B., Abidi, S. H., AbouZeid, O. S., Abraham, N. L., Abramowicz, H., et al. (2019). *The European Physical Journal C, 79*(2). https://doi.org/10.1140/epjc/s10052-019-6540-y
140. Asadi, P., Buckley, M. R., DiFranzo, A., Monteux, A., & Shih, D. (2017). *Journal of High Energy Physics, 2017*(11), 194. https://doi.org/10.1007/JHEP11(2017)194
141. D'Agnolo, R. T., & Wulzer, A. (2019). *Physical Review D, 99*(1), 015014.
142. De Simone, A., & Jacques, T. (2019). *The European Physical Journal C, 79*(4), 289.
143. Sugiyama, M., Suzuki, T., & Kanamori, T. (2012). *Density Ratio Estimation in Machine Learning*. Cambridge University Press. https://doi.org/10.1017/CBO9781139035613
144. Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., & Liao, Q. (2017). *International Journal of Automation and Computing, 14*, 1. https://doi.org/10.1007/s11633-017-1054-2
145. Bach, F. (2017). *Journal Machine Learning Resources, 18*(1), 629.
146. Montanelli, H., & Du, Q. (2017). https://arxiv.org/abs/1712.08688
147. Schilling, M. F. (1986). *Journal of the American Statistical Association 81*(395), 799. https://doi.org/10.1080/01621459.1986.10478337, https://www.tandfonline.com/doi/abs/10.1080/01621459.1986.10478337
148. Henze, N. (1988). *The Annals of Statistics, 16*(2), 772.
149. Wang, Q., Kulkarni, S. R., & Verdu, S. (2005). *IEEE Transactions on Information Theory, 51*(9), 3064. https://doi.org/10.1109/TIT.2005.853314
150. Wang, Q., Kulkarni, S. R., & Verdu, S. (2006). *2006 IEEE International Symposium on Information Theory* (pp. 242–246). https://doi.org/10.1109/ISIT.2006.261842
151. Dasu, T., Krishnan, S., Venkatasubramanian, S., & Yi, K. (2006). *Proceeding Symposium on the Interface of Statistics, Computing Science, and Applications*.
152. Pérez-cruz, F. (2008). *Proceedings of IEEE International Symposium on Information Theory* (pp. 1666–1670).
153. Kremer, J., Gieseke, F., Steenstrup Pedersen, K., & Igel, C. (2015). *Astronomy and Computing, 12*, 67. https://doi.org/10.1016/j.ascom.2015.06.005
154. Qing Wang, S. R. Kulkarni, & Verdu, S. (2005). *IEEE Transactions on Information Theory, 51*(9), 3064. https://doi.org/10.1109/TIT.2005.853314
155. Wang, Q., Kulkarni, S. R., & Verdu, S. (2006). *2006 IEEE International Symposium on Information Theory* (pp. 242–246). https://doi.org/10.1109/ISIT.2006.261842
156. Pérez-cruz, F. (2008). *Proceedings of IEEE International Symposium on Information Theory* (pp. 1666–1670).
157. Manly, B. F. J. (2007). *International Statistical Review, 75*(2), 269. https://doi.org/10.1111/j.1751-5823.2007.00015_21.x, https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1751-5823.2007.00015_21.x._eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1751-5823.2007.00015_21.x
158. Vaart, A. W. V. D. (1998). Cambridge series in statistical and probabilistic mathematics. In*Asymptotic Statistics*. Cambridge University Press. https://doi.org/10.1017/CBO9780511802256
159. Bell, J. S. (1964). *Physics Physique Fizika, 1*(3), 195. https://doi.org/10.1103/PhysicsPhysiqueFizika.1.195, https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195. Publisher: American Physical Society
160. Tishby, N., Pereira, F. C. & Bialek W. (2000). http://arxiv.org/abs/physics/0004057
161. Shwartz-Ziv, R., & Tishby, N. (2017). http://arxiv.org/abs/1703.00810
162. Bahdanau, D., Cho, K., & Bengio, Y. (2015). https://arxiv.org/abs/1409.0473

163. Firat, O., Cho, K., Sankaran, B., Yarman Vural, F. T., & Bengio, Y. (2017). *Computer Speech Languge*, *45*(C), 236–252. https://doi.org/10.1016/j.csl.2016.10.006, https://doi.org/10.1016/j.csl.2016.10.006
164. Celikyilmaz, A., Bosselut, A., He, X. & Choi, Y. (2018). *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (pp. 1662–1675). Association for Computational Linguistics, New Orleans, Louisiana. https://doi.org/10.18653/v1/N18-1150. https://www.aclweb.org/anthology/N18-1150
165. Koopman, B. O. (1931). *Proceedings of the National Academy of Sciences, 17*(5), 315.
166. Koopman, B. O., & Neumann, J. V. (1932) *Proceedings of the National Academy of Sciences, 18*(3), 255. https://doi.org/10.1073/pnas.18.3.255, https://www.pnas.org/content/18/3/255. Publisher: National Academy of Sciences Section: Mathematics
167. Yeung, E., Kundu, S., & Hodas, N. (2017). http://arxiv.org/abs/1708.06850
168. Wehmeyer, C., & Noé, F. (2018). *The Journal of Chemical Physics, 148*(24), 241703.
169. Mardt, A., Pasquali, L., Wu, H., & Noé, F. (2018). *Nature Communications, 9*(1), 5.
170. Takeishi, N., Kawahara, Y., & Yairi, T. (2017) Learning Koopman invariant subspaces for dynamic mode decomposition. http://arxiv.org/abs/1710.04340
171. Otto, S. E., & Rowley, C. W. (2017). http://arxiv.org/abs/1712.01378
172. Li, Q., Dietrich, F., Bollt, E. M., & Kevrekidis, I. G. (2017) *Chaos: An Interdisciplinary Journal of Nonlinear Science, 27*(10). https://doi.org/10.1063/1.4993854, http://arxiv.org/abs/1707.00225
173. Champion, K., Lusch, B., Kutz, J. N., & Brunton, S. L. (2019). *Proceedings of the National Academy of Sciences, 116*(45), 22445.
174. Mezić, I., & Banaszuk, A. (2004). *Physica D: Nonlinear Phenomena, 197*(1–2), 101. https://doi.org/10.1016/j.physd.2004.06.015, https://linkinghub.elsevier.com/retrieve/pii/S0167278904002507
175. Mezić, I. (2005). *Nonlinear Dynamics, 41*(1), 309.
176. Mezic, I. (2019). arXiv:1702.07597. http://arxiv.org/abs/1702.07597
177. Budišić, M., Mohr, R., & Mezić, I. (2012). *Chaos: An Interdisciplinary Journal of Nonlinear Science, 22*(4), 047510. https://doi.org/10.1063/1.4772195, https://aip.scitation.org/doi/10.1063/1.4772195. Publisher: American Institute of Physics.
178. Mezić, I. (2013). *Annual Review of Fluid Mechanics, 45*(1), 357. https://doi.org/10.1146/annurev-fluid-011212-140652, http://www.annualreviews.org/doi/10.1146/annurev-fluid-011212-140652
179. Schmid, P. J. (2010). *Journal of Fluid Mechanics, 656*, 5. https://doi.org/10.1017/S0022112010001217, https://www.cambridge.org/core/journals/journal-of-fluid-mechanics/article/dynamic-mode-decomposition-of-numerical-and-experimental-data/AA4C763B525515AD4521A6CC5E10DBD4. Publisher: Cambridge University Press.
180. Rowley, C. W., Mezić, I., Bagheri, S., Schlatter, P., & Henningson, D. S. (2009). *Journal of Fluid Mechanics, 641*, 115. https://doi.org/10.1017/S0022112009992059, https://www.cambridge.org/core/product/identifier/S0022112009992059/type/journal_article
181. Brunton, S. L., Proctor, J. L., Tu, J. H., & Kutz, J. N. (2015). *Journal of Computational Dynamics, 2*(2), 165. https://doi.org/10.3934/jcd.2015002, https://www.aimsciences.org/article/doi/10.3934/jcd.2015002. Company: Journal of Computational Dynamics Distributor: Journal of Computational Dynamics Institution: Journal of Computational Dynamics Label: Journal of Computational Dynamics Publisher: American Institute of Mathematical Sciences.
182. Arbabi, H. (2018).
183. Lusch, B., Kutz, J. N., & Brunton, S. L. (2018). *Nature Communications, 9*, 4950.
184. Kipf, T., Fetaya, E., Wang, K. C., Welling, M., & Zemel, R., (PMLR, 2018). *Proceedings of Machine Learning Research* (Vol. 80, pp. 2688–2697). http://proceedings.mlr.press/v80/kipf18a.html
185. Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., & Kavukcuoglu, K. (2016). *Proceedings of the 30th International Conference on Neural Information Processing Systems* (pp. 4509–4517). http://dl.acm.org/citation.cfm?id=3157382.3157601

186. Zheng, D., Luo, V., Wu, J., & Tenenbaum, J. B. (2018). http://arxiv.org/abs/1807.09244
187. Chang, M. B., Ullman, T., Torralba, A., & Tenenbaum, J. B. (2016). http://arxiv.org/abs/1612.00341
188. Raposo, D., Santoro, A., Barrett, D., Pascanu, R., Lillicrap, T., & Battaglia, P. (2017). http://arxiv.org/abs/1702.05068
189. Hinton, G. E., & Salakhutdinov, R. R. (2006). *Science, 313*(5786), 504. https://doi.org/10.1126/science.1127647
190. Eslami, S. M. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., Reichert, D. P., Buesing, L., Weber, T., Vinyals, O., Rosenbaum, D., Rabinowitz, N., King, H., Hillier, C., Botvinick, M., D. Wierstra, K. Kavukcuoglu, & Hassabis, D. (2018). *Science, 360*(6394), 1204.
191. Hornik, K. (1991). *Neural Networks, 4*(2), 251.
192. Chen, B., Huang, K., Raghupathi, S., Chandratreya, I., Du, Q., & Lipson, H. (2022). *Nature Computational Science, 2*(7), 433.
193. Kingma, D. P., & Welling, M. (2013). https://arxiv.org/abs/1312.6114
194. Elman, J. L. (1990). *Cognitive Science, 14*(2), 179.
195. Liu, S., Yang, N., Li, M., & Zhou, M. (2014). *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1491–1500). Baltimore, Maryland: Association for Computational Linguistics. https://doi.org/10.3115/v1/P14-1140. https://www.aclweb.org/anthology/P14-1140
196. Auli, M., Galley, M., Quirk, C., & Zweig, G. (2013). *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1044–1054). Seattle, Washington, USA: Association for Computational Linguistics. https://www.aclweb.org/anthology/D13-1106
197. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger *Advances in Neural Information Processing Systems 27* (pp. 3104–3112). Curran Associates, Inc. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
198. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). *IEEE Transactions on Neural Networks, 20*(1), 61.
199. Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2017). arXiv:1511.05493. http://arxiv.org/abs/1511.05493
200. Sanchez-Gonzalez, A., Heess, N., Springenberg, J .T., Merel, J., Riedmiller, M., Hadsell, R., & Battaglia, P. (2018). arXiv:1806.01242. http://arxiv.org/abs/1806.01242
201. Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., & Torralba, A. (2019). *ICLR*.
202. Mrowca, D., Zhuang, C., Wang, E., Haber, N., Fei-Fei, L., Tenenbaum, J. B., & Yamins, D. L. K. (2018). arXiv:1806.08047. http://arxiv.org/abs/1806.08047
203. Bapst, V., Keck, T., Grabska-Barwińska, A., Donner, C., Cubuk, E. D., Schoenholz, S. S., Obika, A., Nelson, A. W. R., Back, T., Hassabis, D., & Kohli, P. (2020). *Nature Physics, 16*(4), 448.
204. Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. W. (2020). Learning to simulate complex physics with graph networks. https://arxiv.org/abs/2002.09405
205. Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., & Ho, S. (2020) arXiv:2006.11287, http://arxiv.org/abs/2006.11287
206. Hamilton, W. L., Ying, R., & Leskovec, J. (2018). arXiv:1706.02216, http://arxiv.org/abs/1706.02216
207. Kipf, T. N., & Welling, M. (2017) arXiv:1609.02907. http://arxiv.org/abs/1609.02907
208. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 3391–3401). Curran Associates, Inc. http://papers.nips.cc/paper/6931-deep-sets.pdf
209. Herzig, R., Raboh, M., Chechik, G., Berant, J., & Globerson, A. (2018). arXiv:1802.05451, http://arxiv.org/abs/1802.05451

210. Liu, Z., & Zhou, J. (2020). Synthesis lectures on artificial intelligence and machine. *Learning, 14*(2), 1.
211. Maddison, C. J., Mnih, A., & Teh, Y. W. (2017). arXiv:1611.00712, http://arxiv.org/abs/1611.00712
212. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/
213. Ross, B. C. (2014). *PLOS ONE, 9*(2), e87357.
214. Kraskov, A., Stoegbauer, H., & Grassberger, P. (2004). *Physical Review E, 69*(6), 066138.
215. Kozachenko, L. F., & Leonenko, N. N. (1987). *Problems Inform. Transmission, 23*, 95.
216. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). *Journal of Machine Learning Research, 12*, 2825.
217. Beléndez, A., Pascual, C., Méndez, D. I., Beléndez, T., & Neipp, C. (2007). *Revista Brasileira de Ensino de Física, 29*(4), 645.
218. This figure is composed of figures in "B. Lusch, J. N. Kutz, and S. L. Brunton, Deep learning for universal linear embeddings of nonlinear dynamics, Nat. Commun. 9, 4950, https://doi.org/10.1038/s41467-018-07210-0, 2018". The work is published under the terms of the Creative Commons Attribution 4.0 license (https://creativecommons.org/licenses/by/4.0/). The labels in the image are modified to fit the context of this book.
219. Paris, M., & Reháček, J. (eds.), (2004). Lecture notes in physics. In *Quantum state estimation*. Berlin, Heidelberg: Springer. https://doi.org/10.1007/b98673. http://link.springer.com/10.1007/b98673
220. Gamel, O. (2016). *Physical Review A, 93*(6). https://doi.org/10.1103/physreva.93.062320, http://dx.doi.org/10.1103/PhysRevA.93.062320
221. Floryan, D., & Graham, M. D. (2022). *Nature Machine Intelligence, 4*(12), 1113.
222. Carleo, G., & Troyer, M. (2017). *Science, 355*(6325), 602.
223. Rocchetto, A., Grant, E., Strelchuk, S., Carleo, G., & Severini, S. (2018). *NPJ Quantum Information, 4*(1), 28. https://doi.org/10.1038/s41534-018-0077-z, https://www.nature.com/articles/s41534-018-0077-z
224. Glasser, I., Pancotti, N., August, M., Rodriguez, I. D., & Cirac, J. I. (2018). *Physical Review X, 8*(1), 011006.
225. Carleo, G., Nomura, Y., & Imada, M. (2018). *Nature Communications, 9*(1), 5322.
226. Cai, Z., & Liu, J. (2018). *Physical Review B 97*(3). https://doi.org/10.1103/PhysRevB.97.035116, http://arxiv.org/abs/1704.05148
227. Huang, Y., & Moore, J. E., (2017). http://arxiv.org/abs/1701.06246
228. Deng, D. L., Li, X., & Sarma, S. D. (2017). *Physical Review B, 96*(19). https://doi.org/10.1103/PhysRevB.96.195145. http://arxiv.org/abs/1609.09060
229. Schmitt, M., & Heyl, M. (2018). *SciPost Physics, 4*(2). https://doi.org/10.21468/SciPostPhys.4.2.013, http://arxiv.org/abs/1707.06656
230. Torlai, G., Mazzola, G., Carrasquilla, J., Troyer, M., Melko, R., & Carleo, G. (2018). *Nature Physics, 14*(5), 447.
231. Nomura, Y., Darmawan, A. S., Yamaji, Y., & Imada, M. (2017). *Physical Review B, 96*(20). https://doi.org/10.1103/PhysRevB.96.205152. http://arxiv.org/abs/1709.06475
232. Deng, D. L., Li, X., & Sarma, S. D. (2017). *Physical Review X, 7*(2). https://doi.org/10.1103/PhysRevX.7.021021. http://arxiv.org/abs/1701.04844
233. Gao, X., & Duan, L. M. (2017). *Nature Communications, 8*(1).https://doi.org/10.1038/s41467-017-00705-2, http://arxiv.org/abs/1701.05039
234. Carrasquilla, J., Torlai, G., Melko, R. G., & Aolita, L. (2019). *Nature Machine Intelligence, 1*(3), 155.

235. Beach, M. J. S., De Vlugt, I., Golubeva, A., Huembeli, P., Kulchytskyy, B., Luo, X., Melko, R.G., Merali, E., & Torlai, G. (2019). *SciPost Physics, 7*(1), 009. https://doi.org/10.21468/SciPostPhys.7.1.009, http://arxiv.org/abs/1812.09329
236. Torlai, G., & Melko, R. G. (2019). http://arxiv.org/abs/1905.04312
237. We use material from "Hendrik Poulsen Nautrup, Tony Metger, Raban Iten, Sofiene Jerbi, Lea M. Trenkwalder, Henrik Wilming, Hans J. Briegel, and Renato Renner, Operationally meaningful representations of physical systems in neural networks, Mach. Learn.: Sci. Technol. 3 045025, https://doi.org/10.1088/2632-2153/ac9ae8, 2022". The work is published under the terms of the Creative Commons Attribution 4.0 license (https://creativecommons.org/licenses/by/4.0/). The material was modified to fit the context of this book (but no essential changes were made to the content).
238. Celikyilmaz, A., Bosselut, A., He, X., & Choi, Y. (2018). arXiv:1803.10357. http://arxiv.org/abs/1803.10357
239. Goldstein, H., Poole, C., & Safko, J. (2002). *Classical mechanics* (3rd edn). Addison-Wesley.
240. González, E. (2020). Master thesis, ETH.
241. Lesort, T., Díaz-Rodríguez, N., Goudou, J. F., & Filliat, D. (2018). *Neural Networks, 108*, 379.
242. Bengio, E., Thomas, V., Pineau, J., Precup, D., & Bengio, Y. (2017). http://arxiv.org/abs/1703.07718
243. Jonschkowski, R., & Brock, O. (2015). *Autonomous Robots, 39*(3), 407.
244. Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2017). *5th International Conference on Learning Representations, ICLR 2017, Toulon, France*, April 24–26, 2017, Conference Track Proceedings. https://openreview.net/forum?id=SJ6yPD5xg
245. Veeriah, V., Hessel, M., Xu, Z., Lewis, R., Rajendran, J., Oh, J., van Hasselt, H., Silver, D., & Singh, S. (2019). arXiv:1909.04607, http://arxiv.org/abs/1909.04607
246. Fan, F., Xiong, J., & Wang, G. (2020). arXiv:2001.02522, http://arxiv.org/abs/2001.02522
247. Inceptionism: Going Deeper into Neural Networks. http://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html
248. Schindler, F., Regnault, N., & Neupert, T. (2017). *Physical Review B, 95*(24), 245134.
249. Seif, A., Hafezi, M., & Jarzynski, C. (2020). *Nature Physics,* 1–9 (2020). https://doi.org/10.1038/s41567-020-1018-2. https://www.nature.com/articles/s41567-020-1018-2. Publisher: Nature Publishing Group.
250. Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., & Soljačić, M. (2019). https://arxiv.org/abs/1912.04825
251. Martius, G., & Lampert, C. H. (2016). arXiv:1610.02995, http://arxiv.org/abs/1610.02995
252. Sahoo, S. S., Lampert, C. H., & Martius, G. (2018). arXiv:1806.07259, http://arxiv.org/abs/1806.07259
253. Lemos, P., Jeffrey, N., Cranmer, M., Ho, S., & Battaglia, P. (2022). Rediscovering orbital mechanics with machine learning. https://doi.org/10.48550/ARXIV.2202.02306. https://arxiv.org/abs/2202.02306
254. Müller, V. C. (2016). *Fundamental issues of artificial intelligence*. Springer.
255. Grace, K., Salvatier, J., Dafoe, A., Zhang, B., & Evans, O. (2018). *Journal of Artificial Intelligence Research, 62*, 729. https://doi.org/10.1613/jair.1.11222, https://www.jair.org/index.php/jair/article/view/11222
256. King, R. D., Whelan, K. E., Jones, F. M., Reiser, P. G. K., Bryant, C. H., Muggleton, S. H., Kell, D. B., & Oliver, S. G. (2004). *Nature, 427*(6971), 247.
257. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). *Proceedings of the IEEE, 109*(1), 43. https://doi.org/10.1109/JPROC.2020.3004555
258. Siegelmann, H., & Sontag, E. (1995). *Journal of Computer and System Sciences, 50*(1), 132. https://doi.org/10.1006/jcss.1995.1013, https://www.sciencedirect.com/science/article/pii/S0022000085710136
259. Graves, A., Wayne, G., & Danihelka, I. (2014). *Neural turing machines*.

260. Lee, J. (2012). Graduate texts in mathematics . In *Introduction to smooth manifolds* (2nd edn). Springer-Verlag. https://doi.org/10.1007/978-1-4419-9982-5, www.springer.com/de/book/9781441999818
261. Achille, A., & Soatto, S. (2018). *IEEE Transactions on Pattern Analysis and Machine Intelligence, 8828*, 1. https://doi.org/10.1109/TPAMI.2017.2784440
262. https://github.com/eth-nn-physics/nn_physical_concepts/blob/copernicus/analysis/copernicus_analysis_elliptic.ipynb
263. https://github.com/eth-nn-physics/nn_physical_concepts
264. https://github.com/tonymetger/communicating_scinet
265. Pitelis, N., Russell, C., & Agapito, L. (2013). *IEEE Conference on Computer Vision and Pattern Recognition,* (pp. 1642–1649). https://doi.org/10.1109/CVPR.2013.215
266. Korman, E. O. (2018). http://arxiv.org/abs/1803.00156
267. Tong, D. http://www.damtp.cam.ac.uk/user/tong/relativity.html