Neural networks and quantum many-body physics: exploring reciprocal benefits

by

Anna Golubeva

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Physics

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Paul Ginsparg Professor, Physics and Information Science, Cornell University
Supervisor(s):	Roger G. Melko Professor, Dept. of Physics & Astronomy, University of Waterloo Associate Faculty, Perimeter Institute for Theoretical Physics
Internal Member:	Anton Burkov Professor, Dept. of Physics & Astronomy, University of Waterloo
Internal-External Member:	Pierre-Nicholas Roy Professor, Dept. of Chemistry, University of Waterloo
Other Member(s):	Michel Gingras Professor, Dept. of Physics & Astronomy, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

I am the sole author of Chapters 1, 2, 4 and 8. Other parts of this thesis consist in part of four manuscripts written for publication in collaboration with others. My contributions to each of these works are detailed in the following.

Research presented in Chapter 3 was a project I worked on during my internship as student researcher at Blueshift, Alphabet Inc (formerly Google X). The co-authors, Guy Gur-Ari and Behnam Neyshabur, were my advisors and collaborators on this project. The formulation of the research question and study design was done jointly. Project execution was done primarily by me, including coding, experiment design and evaluation. My advisors met with me frequently to discuss the results and advise on further steps. Behnam Neyshabur provided technical support with several experiments. Guy Gur-Ari lead the theoretical analysis and conceived the key idea; I did part of the theoretical calculations under his guidance. I drafted the manuscript, my collaborators critically revised several iterations of the manuscript and helped improve the presentation.

Anna Golubeva, Guy Gur-Ari, and Behnam Neyshabur. Are wider nets better given the same number of parameters? In International Conference on Learning Representations, 2021.

Research presented in Chapter 5 was a software project conducted by members of the PIQuIL. I have contributed to the conceptual design of the software package, did code review and testing in the process of writing and suggested improvements, and participating in the writing of the associated manuscript. I was strongly involved in discussions related to RBM theory and wrote a theory-based introduction to RBMs that was published online at https://qucumber.readthedocs.io/en/stable/theory.html as part of the QuCumber package documentation and constitutes Chapter 4 of this thesis.

Matthew J. S. Beach, Isaac De Vlugt, Anna Golubeva, Patrick Huembeli, Bohdan Kulchytskyy, Xiuzhe Luo, Roger G. Melko, Ejaaz Merali, and Giacomo Torlai. QuCumber: wavefunction reconstruction with neural networks. SciPost Phys., 7:9, 2019a. doi: 10.21468/SciPostPhys.7.1.009.

Research presented in Chapter 6 was a project primarily lead by Dan Sehayek. I contributed to the conceptual design of the study and lead the project part related to parameter reduction presented in section III C. For this part, I did the conceptual design, coding, experiment execution and evaluation. I had frequent discussions with Dan and other collaborators throughout the project, provided guidance, assisted with result evaluation and participated in the writing and preparation of the manuscript for publication.

Dan Sehayek, Anna Golubeva, Michael S. Albergo, Bohdan Kulchytskyy, Giacomo Torlai, and Roger G. Melko. Learnability scaling of quantum states: Restricted boltzmann machines. Phys. Rev. B, 100:195125, Nov 2019. doi: 10.1103/PhysRevB.100.195125.

Research presented in Chapter 7 was a project lead by me and advised by Roger Melko. I designed the study, wrote code, conducted the experiments, evaluated results and drafted the manuscript. Roger Melko provided feedback throughout the project, suggested additional experiments, provided intellectual input on manuscript drafts and helped improve the presentation.

Abstract

One of the main reasons why the physics of quantum many-body systems is hard lies in the curse of dimensionality: The number of states of such systems increases exponentially with the number of degrees of freedom involved. As a result, computations for realistic systems become intractable, and even numerical methods are limited to comparably small system sizes. Many efforts in modern physics research are therefore concerned with finding efficient representations of quantum states and clever approximations schemes that would allow them to characterize physical systems of interest.

Meanwhile, Deep Learning (DL) has solved many non-scientific problems that have been unaccessible to conventional methods for a similar reason. The concept underlying DL is to extract knowledge from data by identifying patterns and regularities. The remarkable success of DL has excited many physicists about the prospect of leveraging its power to solve intractable problems in physics. At the same time, DL turned out to be an interesting complex many-body problem in itself. In contrast to its widespread empirical applications, the theoretical foundation of DL is strongly underdeveloped. In particular, as long as its decision-making process and result interpretability remain opaque, DL can not claim the status of a scientific tool.

In this thesis, I explore the interface between DL and quantum many-body physics, and investigate DL both as a tool and as a subject of study. The first project presented here is a theory-based study of a fundamental open question about the role of width and the number of parameters in deep neural networks. In this work, we consider a DL setup for the image recognition task on standard benchmarking datasets. We combine controlled experiments with a theoretical analysis, including analytical calculations for a toy model. The other three works focus on the application of Restricted Boltzmann Machines as generative models for the task of wavefunction reconstruction from measurement data on a quantum many-body system. First, we implement this approach as a software package, making it available as a tool for experimentalists. Following the idea that physics problems can be used to characterize DL tools, we then use our extensive knowledge of this setup to conduct a systematic study of how the RBM complexity scales with the complexity of the physical system. Finally, in a follow-up study we focus on the effects of parameter pruning techniques on the RBM and its scaling behavior.

Acknowledgements

My PhD was the best part of my academic career so far, and I wish to express my sincere gratitude towards the people and organizations who made it such a delightful experience:

My supervisor Roger Melko for supporting all my endeavors and for providing valuable guidance.

The PIQuIL collective I had the greatest pleasure working with and proudly being a member of.

Juan Carrasquilla, whose work was inspiring to me since the beginning of my PhD, for scientific advice and support, and for sharing his immense knowledge.

Angus Galloway for guidinig me into the Machine Learning community, for fun collaborations on adversarial attacks and our joint attempts to understand Deep Learning through information theory.

NSERC, Vector Institute and Borealis AI for funding my research – without it my PhD time could not have been as focussed and enjoyable.

Blueshift, and in particular *Guy Gur-Ari* and *Behnam Neyshabur*, for hosting me in 2019/2020 as an intern.

Perimeter Institute – arguably the best place in the world for a physicist – for hosting me and all the inspiring people I met there.

Thank you all for being part of this journey – it was fun!

Dedication

I dedicate this to the Pickle.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of *Artificial Intelligence* (AI) has long been attracting the interest of physicists, but the advent of **Deep Learning** (DL) took the interactions between physics and AI to a new level. The central paradigm of modern AI is *machine learning* (ML) – a set of techniques that enable a machine (i.e., a computer) to solve complex tasks not by following predefined rules, but instead by deriving the rules by itself based on provided *training data*. DL is technically a subset of ML, comprising learning methods based on *deep* artificial neural networks (DNNs) that came into play only recently; however, the terms ML and DL are often used interchangeably.

The development of these methods propelled ML to a powerful and versatile tool for automation that had a transformative impact on various technological fields. These novel DNN-based tools have enabled machines to solve tasks that were inaccessible with previous methods. For instance, DNNs achieve super-human performance on image recognition (Krizhevsky et al., 2012) tasks, which is the key technology behind autonomous driving. Similar progress was made in speech recognition (Graves et al., 2013) and natural language processing (Collobert et al., 2011; Sarikaya et al., 2014), which have facilitated the development of virtual assistants on our mobile phones and have significantly improved machine translation (Wu et al., 2016).

Given the remarkable progress that DL has elicited in industry, it became enticing to scientists to apply DL to intractably difficult problems that can not be easily solved with conventional methods. Ultimately, such problems in science are intractable for a similar reason as those tasks in industry at which DL excels: They involve high-dimensional spaces and large datasets. Therefore, it appears intuitive that DL could be the right tool to use. Among the earliest attempts to deploy DL in physics were the search for exotic particles

in high-energy physics (Baldi et al., 2014) and efficient solution of dynamical mean-field theory with ML (Arsenault et al., 2015). Finally, Carrasquilla and Melko (2017) applied ML to classify phases of quantum matter and caused a cascade of further works, such that within the following three years a new scientific field has formed. Similar developments happened in other subfields of physics, including quantum computing, particle physics and cosmology, physical chemistry and material science. A recent review article by Carleo et al. (2019) surveys the application areas and successes of DL in these domains.

However, the success of DL is being eyed with suspicion, because the tool itself is largely not understood. The development of DL has been driven almost exclusively by the purpose of non-scientific industrial applications, which naturally emphasize fast progress rather than careful understanding. Successful practices and design principles for DL systems[1] have been obtained by trial and error. Fueled by impressive achievements, the empirical advancement of DL continued to rush forward, while the theory was left behind. Indeed, in many non-scientific use cases DL tools can be highly effective and useful without a theoretical foundation. However, the lack of interpretability and many instances of unexplained behavior are putting limits on the applicability of DL tools. Deployment of DL in science – or any setting that requires a justification of automated decisions and a reliable uncertainty estimation – demands an understanding that goes beyond the empirically collected knowledge and ad-hoc heuristics. Thus, in order to establish DL as a tool for scientific problems, scientists have to solve the problem of DL first.

The reason for the current underdeveloped state of DL theory, however, is not pure neglect. DL experiments have been made maximally accessible through numerous software libraries (Paszke et al., 2019; Abadi et al., 2015; Chollet et al., 2015; Jia et al., 2014) that essentially provide DL construction kits with all required algorithms and training pipelines already built-in, thus allowing anyone to build rather powerful DL systems on a personal computer without much knowledge. In contrast, a formal theoretical treatment of such systems is forbidding and presents a hard challenge for scientists of any domain. The reason why DL systems are so hard to analyze lies in the following:

(1) The sheer number of interacting parts in the system and the dependence between them.

(2) The stochasticity present in these systems.

(3) The nature of empirical datasets that these learning systems are trained on.

[1]Here, "DL system" refers to all parts of a DL setup for a given learning task, including the neural network, its architecture, the training protocol, dataset etc.

Point (1) pertains both to the learning *system* and to the learning *process*, including such aspects as model architecture, learning algorithm, and training procedure. Point (3) addresses the fact that DNNs are typically trained on datasets that are not well characterized. For instance, natural images or text samples collected automatically from the internet are common DL training sets. Due to their enormous size[2], a human review of such datasets is not feasible, and as a result many important characteristics of the dataset remain unknown. This is problematic because data that is used to train the learning algorithm largely determines the properties of the resulting DL tool.

Remarkably, points (1) and (2) are essentially the same difficulties as we face in statistical and quantum many-body physics. In fact, artificial neural networks, which are at the heart of DL, can be seen as valid examples of interacting many-body systems. Excited by this analogy, some physicists have flipped the research question and turned towards DL systems as the subject of study, attempting to gain insights into the fundamental principles of these systems with the tools of theoretical physics.

The oldest strand of research of this kind studies NN-based learning systems through the lens of statistical mechanics, or more precisely the *statistical physics of disordered systems* (Engel and Van den Broeck, 2001). Central to this approach is the concept of a *spin glass* – a disordered dynamical system characterized by randomness. Spin glasses display features that are commonly found in various complex systems, including biological and artificial neural networks (Amit et al., 1985; Watkin et al., 1993). Therefore, the set of mathematical tools developed for spin glasses, based on the replica and cavity methods, and the related message passing algorithms, can be used to analyze such systems. In the 1980s-1990s these methods have facilitated theoretical progress in the understanding of NNs. Unfortunately, however, they can only describe elementary NN structures, and thus the explanatory power of obtained results is limited to very simple learning machines. There have been efforts to extend these methods to DL, e.g. (Choromanska et al., 2015; Zdeborová and Krzakala, 2016), yet the vast empirical evidence indicates that the essence of DNNs can not be captured by describing the sum of their building blocks.

A plethora of work is dedicated to examining analogies between physics and various aspects of DL. For instance, the hierarchical manner in which a multi-layer NN processes input data is reminiscent of a coarse-graining procedure. This observation has led to speculations that DL might be related to the renormalization group (RG) flow. The connection between RG and DL was studied in a number of works (Bény, 2013; Mehta and Schwab, 2014; Iso et al., 2018; Koch-Janusz and Ringel, 2018; Li and Wang, 2018; Funai and Giata-

[2]Among the current standard datasets used for benchmarking DL systems, the smaller ones comprise about 60 thousand instances, and the larger ones have over a million instances.

ganas, 2020; De Mello Koch et al., 2020) that have found interesting parallels. However, a recent strand of research has put the correspondence between DNNs and Quantum Field Theories (QFT) on a solid mathematical foundation (Dyer and Gur-Ari, 2019; Cohen et al., 2021; Antognini, 2019; Yaida, 2020; Halverson et al., 2021), making it evident that the relationship between DL and RG is rather a "spiritual resemblance", but not an exact equivalence. The connection between DNNs and QFT is based on the fact that infinitely large DNNs are drawn from Gaussian processes (Lee et al., 2018a; de G. Matthews et al., 2018), the analog of non-interacting field theories. Other examples of physics-inspired studies have exploited symmetries in the spirit of Noether's theorem to analytically describe the learning dynamics of DL systems (Kunin et al., 2021); mean-field techniques to characterize the function of particular elements or structures in DNNs (Yang and Schoenholz, 2017; Yang et al., 2019; Gilboa et al., 2019); and power-law scaling (Bahri et al., 2021) to explain empirical observations of scaling behavior in modern DNNs (Kaplan et al., 2020). More examples can be found in a recent review by Bahri et al. (2020).

In this thesis, I present several studies that are situated at the intersection between quantum many-body physics and DL. In my research, I explore both directions: How can DL be leveraged as a tool to solve problems in physics and how can the principles of theoretical physics be harnessed to advance our understanding of DL systems.

Outline of the Thesis

Chapters 3, 5, 6 and 7 present my contributions to the field in form of research projects that I have worked on with various collaborators.

Chapter 1 is the current chapter, providing an introduction and motivating the theoretical study of DL, in particular from the perspective of a physicist. In chapter 2 I describe in more detail what kind of challenges a theoretical treatment of DL faces. I then provide a mathematical introduction to the central concepts of DL, as well as methods of DL theory that will play a role in subsequent chapters. Chapter 3 presents a study aimed at understanding a specific empirical phenomenon in DL through a combination of targeted experiments and theory. In chapter 4 I review the theory of Restricted Boltzmann Machines (RBMs), as the following three chapters feature RBMs as the main tool. Chapter 5 is about the deployment of RBMs as generative models for quantum-state reconstruction; chapter 6 systematically studies the scaling of computational resources required for the task of quantum state reconstruction with an RBM; chapter 7 builds on the previous study and examines the scaling behavior of an RBM under the application of a parameter reduction technique called pruning. I conclude in chapter 8.

Chapter 2

Theoretical approaches to Deep Learning

Part of this chapter used to be published as an article on the website "physicsml" that went offline in early 2021.

2.1 Deep Learning challenges classical learning theory

Why do modern deep neural networks (DNNs) perform so well on previously unseen test data, even when their number of weights is much larger than the number of data points in the training set? This question keeps puzzling many theorists and practitioners doing Deep Learning (DL), in particular those who are used to the rules of classical statistical modeling. For example, consider the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015), which became a benchmark for state-of-the-art DNNs. ImageNet is a database of labeled high-resolution images, designed for the purpose of testing visual object recognition software. In the annual ILSVR challenge, ever new and sophisticated DL models compete in classifying the data set of 1.2 million images from 1000 categories. Among the leaders on the winners list we find GoogLeNet, AlexNet and VGGNet, with 4, 60 or 138 millions parameters, respectively.

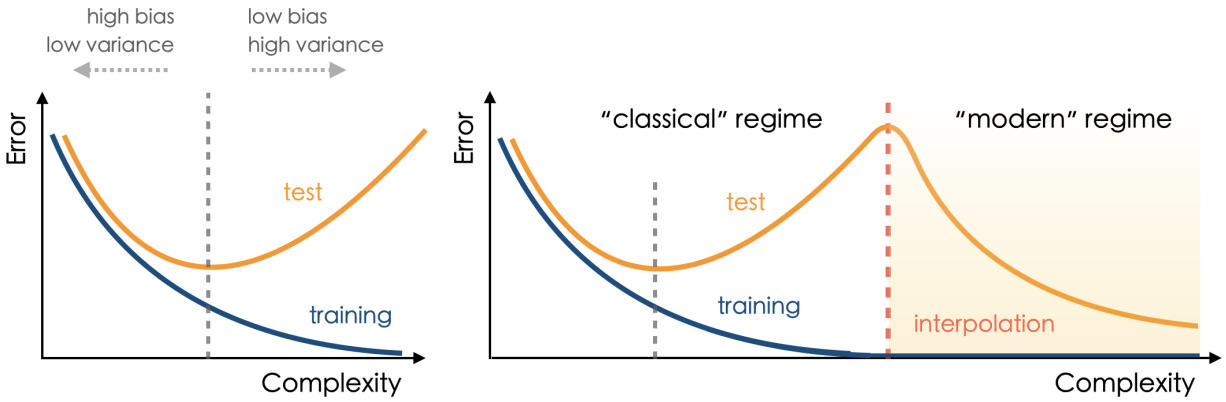
This degree of *overparametrization* is insane from the point of view of classical statistical learning theory, which teaches us parsimony in model size selection. The argument for this is intuitive: A larger number of parameters increases the model's flexibility, allowing it to express more complex functions. However, a more flexible model will adjust to many

details of the training set which are not the general features of the data. As a result, its performance on a previously unseen test data will be poorer. To sum up, the larger the number of parameters in a model, the more prone it is to "overfitting".

This is known as the **bias-variance trade-off**, a central concept in statistical learning theory used to characterize the performance of parametric models. In this context, the variance refers to the amount by which the fit function changes if estimated on a different training set, and the bias measures how much the chosen ansatz for the fit function is off from reality. Generally, when the model flexibility is increased, the variance will increase and the bias will decrease. Figure 2.1a illustrates these concepts. The art of achieving a fit that generalizes well lies in selecting a model that is sufficiently complex to capture the reality and still rigid enough to ignore non-general features.

The discrepancy between the training and the test error is called the **generalization gap**, and is used as a measure for the generalization ability of a model. In statistical learning theory, models are typically characterized through a bound on the generalization gap, which is derived based on some notion of model complexity. A complexity measure is not uniquely defined; there exists a variety of different suggestions, such as the *Rademacher complexity* or the *Vapnik-Chervonenkis (VC) dimension*. The latter, for instance, is expected to be proportional to the number of parameters in the model. A classic result states that in leading order the generalization gap scales as $GG \propto \sqrt{M/N}$, where M is a measure for the model complexity, and N is the number of examples in the training set. Thus, to put it simply, the model is not expected to generalize well as long as the number of parameters exceeds the size of the training data set. The use of highly overparametrized models in DL seems to directly contradict this rule in practice: Large DNNs commonly achieve near-zero training error on data sets of moderate size, while still having excellent performance on the test set. Moreover, the smallest test error is typically achieved by the largest model.

The complexity of neural networks, particularly the DNNs, and the scale of modern data sets, makes them essentially intractable. While a theoretical characterization of the model complexity in modern DL remains elusive, many insights are gained from targeted experimental analyzes. For instance, Zhang et al. (2017a) is a famous empirical work that has pinned down the extent of DNN capabilities by establishing that modern DNNs have enough capacity to fit random noise. They proved this claim in a series of simple and systematic experiments on state-of-the art DNNs for the image classification task. Essentially, they have demonstrated that a DNN that shows good generalization ability on the original data set, still achieves zero training error when the original labels in the data set are replaced by random labels, while the test error, of course, remains at the level of random guessing. Moreover, even in the case when the true images in the original data



(a) classical bias-variance curve (b) modern extension with "double descent"

Figure 2.1: Illustration of the bias-variance trade-off: Prediction error on the training (blue) or test (orange) sample as a function of model complexity. (a) Is the classical U-shaped curve from statistical learning theory. Dashed vertical line indicates the sweet spot between "underfitting" (model complexity not large enough to capture the modeled relations) and "overfitting" (model is too flexible, it is fitting the training data too closely and fails to generalize).

set were replaced by completely random pixels, convolutional DNNs still achieved zero training error. This result implies that a sufficiently large DNN has enough *capacity* to fit training data that does not contain any learnable structure. The key insight provided by these experiments is that the generalization gap of a model can be substantially increased without changing the model, its size, the hyper-parameters or the optimizer, but rather by randomizing the labels alone. This fact challenges the conventional view of statistical learning theory, which establishes bounds on the generalization gap based on some model features, while fully disregarding qualitative properties of the data set.

Because of the numerous seemingly unrecoverable contradictions, many consider statistical learning theory to be plainly the wrong tool for DL. Others, however, advocate for holding on to the old concepts, and attempt to update them to account for the novelty of DL. For instance, Belkin et al. (2019) made an interesting proposal to fit modern DNNs into the bias-variance trade-off. They argued that for DL, the U-curve picture needs to be extended along the complexity axis far beyond the point of zero training error as shown in Figure 2.1b, as typical DNNs are complex enough to fit the training data perfectly. In this *interpolation regime* where the training error is zero, a small test error is recovered when model complexity is increased further – a phenomenon dubbed "double descent", which

has been observed in many typical DL settings.

Motivated by the fact that larger models show better performance, it is natural to wonder what happens when the model size goes to *infinity*. While practitioners are limited by the available compute, theorists have no reservations against such endeavors. Indeed, similar to the situation in statistical physics, taking the number of NN parameters to infinity results in tremendous simplifications and makes DNNs for once tractable. And even though conclusions drawn in the infinite-model limit do not directly transfer to finite-size models in general, they still provide useful guidance and help us advance the theoretical foundations of DL.

2.1.1 Towards a theory of Deep Learning

The size of a NN model is measured by the number of its weights. Letting the number of weights go to infinity allows us to treat them collectively as a statistical ensemble. Specifically, this limit is achieved by taking the NN *width* to infinity, which corresponds to the number of hidden units in a fully-connected layer, or to the number of output channels in a convolutional layer. This **infinite-width limit** has opened up many research directions towards a theoretical characterization of NNs. In this work, I focus mainly on one of these directions which exploits the connection between NNs and **kernel methods**.

Interestingly, the research of NN kernels was developed by contributions from very different approaches. In fact, the infinite-width limit was first considered back when NNs were not deep. The idea was put forward by Neal (1994) who approached NNs from a *Bayesian* perspective. He was the first to draw the connection between NNs and kernels by establishing that in the infinite-width limit single-layer NNs become equivalent to **Gaussian Processes** (GPs) with a specific kernel (i.e., the GP covariance function). Subsequently, Williams (1998) has shown how to compute this kernel analytically. Such infinitely wide NN models are now called *Neural Network Gaussian Processes* (NNGPs). About ten years later, at the beginning of the DL era, DNNs were shown to outperform conventional learning approaches, yet the lack of theoretical foundation for these DL methods caused many reservations. Some researchers attempted to import the advantages of the novel DL methods into the solid framework of conventional techniques. For instance, Cho and Saul (2009) explored the possibility of doing DL in kernel machines: They derived compositional kernels that mimic the computation in DNNs and used these to build "deep kernel machines". Later, this type of kernels proved to be useful in the context of NNGPs again, when Neal's and Williams' results were extended to DNNs with arbitrarily many layers (Lee et al., 2018a; de G. Matthews et al., 2018), and further generalized to a va-

riety of nonlinearities and architectures, (Garriga-Alonso et al., 2019; Novak et al., 2020; de G. Matthews et al., 2017).

The connection between Gaussian Processes (GPs) and Kernel methods (KMs) is subtle. Despite the fact that GPs involve a kernel, they are not an instance of KMs. In particular in machine learning context, GPs and KMs are both nonparametric approaches based on positive-definite kernels, widely used for modeling nonlinear functional relationships. The fundamental difference between them is the following: GPs are a method of *Bayesian* ML – it models a problem probabilistically and produces a posterior distribution for the unknown function of interest. KMs are *frequentist* models with a decision-theoretic approach to learning – that is, they solve the task by defining a loss function and optimizing the empirical risk. Kanagawa et al. (2018) dedicate a comprehensive review paper to the connections and equivalences between these methods. In the context of my work presented here, these details are not relevant, and thus I will be providing only a brief introduction to each of these methods in the following sections. But before getting to the methods, I shall introduce the concept of Deep Learning with artificial neural networks more formally.

2.2 Deep Learning: a formal introduction

In this section I provide a brief introduction to the essential concepts of Deep Learning with artificial neural networks and set up the notation for future chapters. For more mathematical details on various aspects of DL and additional references, consult the current review article by Berner et al. (2021).

Learning

Learning is the process of gaining knowledge from data. In statistical terms, it is the process of *inductive inference*, which can be roughly decomposed into three steps: (1) make observations, (2) construct a model of the underlying phenomenon, and (3) make predictions using this model. **Statistical Learning Theory (SLT)** formalizes learning as the problem of function estimation from a given collection of data. Originally, SLT was concerned with the mathematical analysis of learning models, but eventually it expanded to include the design of learning algorithms that emerged based on the established theoretical foundation. This branch grew into a separate field called **Machine Learning (ML)**, which is essentially focused on the goal of automating the learning process. **Deep Learning (DL)** is a (large and comparably novel) subset of ML which emerged as advances in

ML expanded to learning with *deep neural networks*. Nowadays, ML is heavily dominated by applications, and the developments on the practical side have by far outpaced theory.

The learning problem

According to Vapnik (1995), the model of *learning from examples* can be analyzed in a general statistical framework of minimizing expected loss using observed data.

We consider an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, and assume that the input-output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ are random variables distributed according to an unknown probability distribution $P(x, y) = P(x)P(y|x)$. Further, we assume that the relation between x and y is perfectly captured by some unknown function g as $g(x) = y$, which we call the *target function* (also referred to as *ground truth*). We define an ansatz for the target function as $f_\lambda(x)$ parametrized by $\lambda \in \Lambda$ which returns a prediction $\hat{y} = f_\lambda(x)$ for a given x. This assumption defines the space of possible functions $\mathcal{F} = \{f_\lambda(x)\}$, referred to as the *hypothesis space*. Note that in order to specify $\mathcal{F}$ we need to make an assumption about the form of the target function. For example, if we assume that the target function is linear, then $\mathcal{F}$ can be formally expressed as

$$\mathcal{F} = \{f_\lambda : \mathcal{X} \to \mathcal{Y} \mid f_\lambda(x) = \lambda_1 x + \lambda_2\} ,$$

that is, the set of all linear functions f_λ mapping from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$.

The problem of learning essentially consists in choosing from $\mathcal{F}$ a single hypothesis f_λ which provides the best estimate for the target function g. This selection is based on a **training set** of m random *independent identically distributed* (i.i.d.) observations drawn according to $P(x, y)$:

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_m, y_m)\} . \tag{2.1}$$

The discrepancy between the prediction $\hat{y}$ and the target output y for a given input x is measured by a **loss** $\mathcal{L}(y, \hat{y})$. The *expected value* of this loss is called the **risk functional**:

$$R(f_\lambda) = \int dP(x, y)\ \mathcal{L}(y, f_\lambda(x)) = \mathbb{E}_{(x,y)\sim P(x,y)} \left[\mathcal{L}(y, f_\lambda(x))\right] . \tag{2.2}$$

Thus, the solution to the learning problem consists in finding in the set $\mathcal{F}$ the function

$$f_\lambda^0 = \arg\min_{f \in \mathcal{F}} R(f_\lambda) , \tag{2.3}$$

which minimizes the risk functional (2.2) given an i.i.d. sample (2.1) while the probability measure $P(x, y)$ is unknown. Three main types of learning problems – classification, regression estimation, and density estimation – are specific cases of this general formulation.

Empirical Risk Minimization

In practice, we can not minimize the risk functional directly when the probability measure is unknown. Therefore, we introduce a proxy for the risk functional constructed based on the training set, called the **empirical risk functional**:

$$R_{\text{emp}}(f_\lambda) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y_i, f_\lambda(x_i))\,. \tag{2.4}$$

We then approximate the function $f_\lambda^0(x)$ which minimizes the true risk (2.2) by the function

$$f_\lambda^\star = \arg\min_{f\in\mathcal{F}} R_{\text{emp}}(f_\lambda) \tag{2.5}$$

which minimizes the empirical risk (2.4). This approach is referred to as the **Empirical Risk Minimization (ERM)** principle. In particular, the *least-squares method* in the problem of regression estimation and the *maximum-likelihood method* in the problem of density estimation are well-known instances of the ERM principle.

The minimization is carried out by an *optimization algorithm* of choice. This operation is in general computationally expensive, thus requiring to trade off accuracy for computation cost – for instance, by stopping an iterative optimization algorithm before convergence (Bousquet et al., 2004). Consequently, the result obtained by approximate optimization, denoted as $\tilde{f}_\lambda^\star$, will deviate from the ERM solution $f_\lambda^\star$, such that

$$|R_{\text{emp}}(\tilde{f}_\lambda^\star) - R_{\text{emp}}(f_\lambda^\star)| \le \rho\,, \tag{2.6}$$

where $\rho \ge 0$ is a predefined tolerance.

Error decomposition

The practical solution $\tilde{f}_\lambda^\star$ deviates from the target function g by an error ε which is composed of three parts arising due to approximation, estimation and optimization:

$$\varepsilon := \mathbb{E}\Big[R(\tilde{f}_\lambda^\star) - R(g)\Big] \tag{2.7}$$

$$= \mathbb{E}\big[R(f_\lambda^0) - R(g)\big] + \mathbb{E}\big[R(f_\lambda^\star) - R(f_\lambda^0)\big] + \mathbb{E}\Big[R(\tilde{f}_\lambda^\star) - R(f_\lambda^\star)\Big] \tag{2.8}$$

$$= \varepsilon_{\text{app}} + \varepsilon_{\text{est}} + \varepsilon_{\text{opt}}\,, \tag{2.9}$$

where the expectations are taken with respect to the random choice of the training set.

The **approximation error** measures how closely the best function in $\mathcal{F}$ can approximate the ground truth g. It is determined by the expressivity of the chosen ansatz f_λ and the resulting hypothesis space $\mathcal{F}$. The **estimation error** originates from minimizing the empirical risk instead of the true risk, and can be reduced by increasing the number of examples in the training set. The **optimization error** arises from inaccurate optimization, as discussed above, and can be reduced at the cost of longer computation time.

Types of learning problems

The main three types of learning problems are the following:

Regression. Regression considers real-valued outputs $y \in \mathbb{R}$. The target regression function $f^\star_\lambda(x) = \int y \, dP(y|x)$ is known to be the one which minimizes the mean squared error – that is the functional (2.4) for the *quadratic loss*:

$$\mathcal{L}(y, f_\lambda(x)) = (y - f_\lambda(x))^2 \,. \tag{2.10}$$

Classification. Classification can be viewed as a special case of regression where the output y takes only two values $y \in \{0, 1\}$, and $\mathcal{F}$ is a set of indicator functions. The loss function is called the *0-1 loss*, and is given by:

$$\mathcal{L}(y, f_\lambda(x)) = \begin{cases} 0 & \text{if } y = f_\lambda(x) \\ 1 & \text{if } y \neq f_\lambda(x). \end{cases} \tag{2.11}$$

This kind of learning problem is considered in the study presented in the next chapter 3.

Density estimation. Density estimation problem is typically solved by the *maximum-likelihood estimation* (MLE) method which selects the density function $f^\star_\lambda(x)$ that maximizes the probability of the data in the training sample. The corresponding loss function is called *negative log-likelihood*, and is given by:

$$\mathcal{L}(y, f_\lambda(x)) = -\log f_\lambda(x) \,. \tag{2.12}$$

The density estimation problem is discussed in chapter 4 in the context of Restricted Boltzmann Machines.

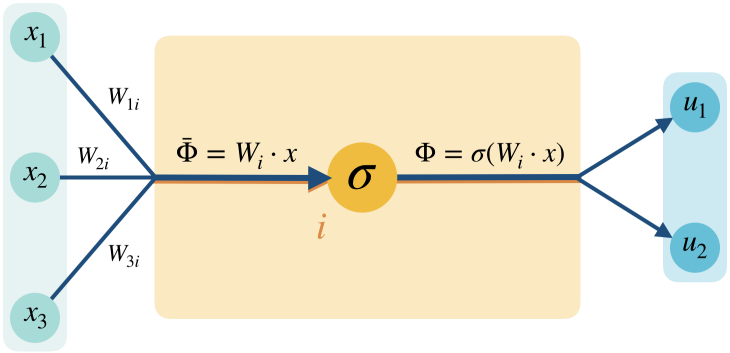


Figure 2.2: Computation in a single neuron of a NN: All incoming signals (i.e., the elements of input vector x) to the i-th neuron in a hidden layer are weighted by the neuron's weights W_i and summed into a single input, called the pre-activation $\bar{\Phi}$. Application of the activation function σ returns the activation Φ, which is passed forward to each neuron of the next layer.

Artificial neural network

An artificial neuron was originally introduced as a simplified model of a biological neuron (McCulloch and Pitts, 1943), and later repurposed as a computational element of a learning machine by Rosenblatt (1958), who proposed the first trainable multi-layered network of such artificial neurons. Neural-network based learning became the centerpiece of ML, and later on a variety of NN architectures was developed empirically. In particular since the advent of DL, NN design has been driven predominantly by the demands of industrial applications, rather than by analogy with biological systems. Below I discuss the *fully-connected feedforward NN* – the basic architecture in ML – in detail. For an overview of NN architectures, visit the NN zoo by Leijnen and Veen (2020).

In general, a NN can be viewed as a graph: Graph vertices represent individual neurons, and the edges indicate weighted connections between the neurons. Each neuron is equipped with an *activation function* σ and an array of weights w for all incoming connections. The NN is structured into *layers*, each associated with a *weight matrix* W, which is composed of the weight vectors of all neurons in the layer. The number of neurons in a layer is the *layer width*, and the total number of layers is referred to as NN *depth*. The NN input is regarded as layer 0, the last layer is referred to as the *output layer*, and all layers inbetween are called *hidden layers*. A NN with more than one hidden layers is considered *deep*. Neurons within a layer typically share the same activation function and are not connected to each other. In the context of a learning problem, a NN is a specific implementation of the function

family $\mathcal{F} = \{f_\lambda(x)\}$ parametrized by the learnable NN parameters λ. Each neuron can be viewed as an elementary computational unit of the NN function. The scheme in Figure 2.2 explains the principle of computation in a NN.

Fully-connected feedforward NN. A fully-connected feedforward NN (FCNN) consists of L layers with N_ℓ neurons in each layer ℓ, and the respective activation functions $\sigma_\ell : \mathbb{R} \to \mathbb{R}$. The width and depth of the architecture are given by $\|N\|_\infty$ and L. The learnable parameters λ of this NN are the weight matrices $W^{(\ell)} \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ for $\ell = 1, \dots, L$ and bias vectors $b^{(\ell)} \in \mathbb{R}^{N_\ell}$ for $\ell = 1, \dots, L-1$. The total number of learnable parameters is given by

$$P = \sum_{\ell=1}^{L} N_\ell N_{\ell-1} + N_\ell \,. \tag{2.13}$$

In practice, biases are optional in most architectures, and in theory, it is common to absorb the bias into the weight vector of the neuron by defining an additional (zeroth) coordinate of the input and setting its value to 1 in order to declutter the notation.

The *NN function* is defined by $f_\lambda(x) := \bar{\Phi}_\lambda^{(L)}(x)$, where the functions $\bar{\Phi}_\lambda^{(\ell)}(\cdot) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_\ell}$ are called the *pre-activations* and $\Phi_\lambda^{(\ell)}(\cdot) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_\ell}$ the *activations* of the ℓ-th layer, and are defined as

$$\bar{\Phi}_\lambda^{(1)}(x) = W^{(1)}x + b^{(1)} \,, \tag{2.14}$$

$$\bar{\Phi}_\lambda^{(\ell+1)}(x) = W^{(\ell+1)}\Phi_\lambda^{(\ell)}(x) + b^{(\ell+1)} \,, \tag{2.15}$$

$$\Phi_\lambda^{(\ell)}(x) = \sigma(\bar{\Phi}_\lambda^{(\ell)}(x)) \,, \tag{2.16}$$

for $\ell = 1, \dots, L$, where σ is applied component-wise. The corresponding function space, a.k.a. the *hypothesis space*, is defined as $\mathcal{F} = \{f_\lambda : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}\}$. The NN *realization function* $F : \mathbb{R}^P \to \mathcal{F}$ maps parameters λ to functions f_λ in the space $\mathcal{F}$.

Common activation functions used in practice are variants of the *rectified linear unit* (ReLU), given by $\sigma(x) := xH(x)$, where $H(x) := \mathbf{1}_{x>0}$ denotes the Heaviside step function, and *sigmoidal* functions, such as the logistic function $\sigma(x) := 1/(1+e^{-x})$, or $\sigma(x) := \tanh(x)$.

More sophisticated architectures may include recurrent connections (loops in the NN graph), parameter sharing, and computational operations other than the weighted sum of inputs. However, the basic concepts explained here still apply and can be generalized to these architectures.

Training

Training a NN refers to the process of optimizing its parameters with the goal of fitting the target function based on the given data, i.e., the training set. Commonly, *gradient-based* optimization algorithms are employed for this task, the most popular in contemporary DL being Stochastic Gradient Descent and its variants (Ruder, 2017).

Gradient Descent (GD) is an iterative method for finding a local minimum $w^\star$ of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$. The principle is intuitive: Starting from a random value $w^{(0)}$, move stepwise into the direction of steepest descent, which is given by the negative gradient of f at the current point w. This procedure is summarized by the following *update rule*:

$$w^{(t+1)} = w^{(t)} - \eta \nabla f(w^{(t)}) \tag{2.17}$$

with step size $\eta > 0$, also called the *learning rate* in the context of ML.

Now consider the case of ERM in a ML setting, where the objective function is the empirical risk (2.4) and the optimization is done with respect to the NN parameters λ. Standard GD performs updates computed on the entire training set containing m examples:

$$\lambda^{(t+1)} = \lambda^{(t)} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla \mathcal{L}(y_i, f_\lambda(x_i)) \,. \tag{2.18}$$

This step is computationally expensive for large training sets. A practical workaround is to perform each update step based on a subset of the training set, called a *mini-batch*. The result is a stochastic approximation to the GD algorithm, known as **Stochastic Gradient Descent (SGD)**. The SGD algorithm can be summarized in pseudo-code as follows:

> Choose an initial parameter vector $\lambda^{(0)}$.
>
> For `epoch = 1...k`, do:
>
> > Randomly shuffle examples in the training set.
> >
> > For `i = 1...b`, do:
> >
> > $\lambda^{(t+1)} = \lambda^{(t)} - \eta \frac{1}{m'} \sum_{i=1}^{m'} \nabla \mathcal{L}(y_i, f_\lambda(x_i))$

Here, m' is the mini-batch size and b is the number of mini-batches, chosen such that $m' \cdot b = m$. The term *epoch* refers to the number of times SGD cycles through the training set. An alternative to setting a fixed number of epochs (here k) is to continue training until a convergence criterion is fulfilled. Note that due to its stochastic nature SGD never stops

– unlike standard GD that stops when the gradient vanishes. That means, convergence of SGD has to be defined via some threshold. The advantage, however, is that SGD can escape saddle points and local maxima, whereas GD gets stuck at any stationary point of the objective function.

2.3 Kernel Methods

Kernel Methods (KMs) are arguably one of the most powerful techniques for nonlinear statistical learning problems. KMs belong to the class of nonparametric learning methods, and usually take a decision-theoretic approach to learning by defining a loss function and optimizing the empirical risk (Kanagawa et al., 2018). The main idea of KMs is to combine the *expressive power* of nonlinear transformations with robust and *simple algorithms* for linear problems. In particular, KMs achieve this by embedding the input data in a (higher-dimensional) vector space called the **feature space**, thus allowing for non-linear representations. Then, a linear model is applied on these features instead of directly on the inputs themselves.

What makes this method fly is the **kernel trick**: Firstly, the algorithms are formulated solely in terms of inner products of inputs, such that the coordinates of the embedded points are not required, only their pairwise inner products. (This procedure is called *kernelization.*) Secondly, the algorithms are formulated such that the pairwise inner products in feature space can be computed directly from the original inputs using a **kernel function** k. The kernel function implicitly defines the mapping into the feature space, and thus has to be chosen dependent on the specific data type and domain knowledge of the data source (Shawe-Taylor and Cristianini, 2004).

To formulate this in mathematical notation, let $x, x' \in \mathcal{X} \subseteq \mathbb{R}^d$ be two data points in the input space $\mathcal{X}$, and $\phi : \mathcal{X} \to \mathcal{H}$ be a nonlinear feature map between the input space $\mathcal{X}$ and the feature space $\mathcal{H}$. The inner product between $\phi(x)$ and $\phi(x')$ in $\mathcal{H}$ can be computed as $\langle \phi(x), \phi(x') \rangle_{\mathcal{H}} = k(x, x')$ using a kernel function $k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. In practice, the kernel function k is directly given in terms of x, x', such that the inner product $\langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$ can be obtained without finding the explicit expression of ϕ.[1]

Kernelization has been used to enhance linear models. A notable example are *Support Vector Machines* (SVMs), which represent one of the most influential early approaches to supervised learning (Boser et al., 1992; Cortes and Vapnik, 1995). Kernel SVMs dominated

[1]For a comprehensive mathematical treatment of KMs refer to Liu et al. (2021) and Shawe-Taylor and Cristianini (2004).

ML until Hinton et al. (2006) demonstrated that NNs can outperform the most powerful kernel SVM on the MNIST benchmark (Goodfellow et al., 2016). The downside of kernelization is that we need to compute the kernel matrix that consists of k applied to all pairs of data points. As a result, a kernelized learning algorithm requires time that grows at least quadratically in the data set size, and predictions with a kernelized procedure require access to the entire training set. For instance, given N samples in the original d-dimensional input space $\mathcal{X}$, kernel ridge regression (a popular KM tool for regression) requires $\mathcal{O}(N^3)$ training time and $\mathcal{O}(N^2)$ space to store the kernel matrix (Liu et al., 2021). Because of the large computation and storage costs, KMs become infeasible for large training sets.

Kernel approximation

The poor scalability of KMs can be overcome via **kernel approximation**. A variety of algorithms has been developed to achieve this, including data-dependent and independent techniques. One of the most popular data-independent techniques is the **Random Features** model proposed by Rahimi and Recht (2008): Instead of leveraging the kernel trick, construct an *explicit* mapping $\psi : \mathbb{R}^d \to \mathbb{R}^n$ to a low-dimensional feature space, such that the inner product between a pair of transformed points approximates their kernel value,

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \approx \psi(x)^\top \psi(x'). \tag{2.19}$$

Note that unlike the kernel feature map ϕ, the randomized feature map ψ is low-dimensional. This approximation results in substantial computational savings, allowing for a linear model to be learned in the transformed space in $\mathcal{O}(Nn^2)$ time and with $\mathcal{O}(Nn)$ memory, while still retaining the expressive power of nonlinear methods. In the original work (Rahimi and Recht, 2008), the authors have demonstrated that RF models even with small dimension n achieve excellent results on regression and classification tasks.

2.4 Neural network kernels

The random feature model can be realized as a two-layer NN where the weights in the first layer are fixed and only the weights in the output layer are trained. Specifically, consider a two-layer NN with parameters $u \in \mathbb{R}^{n \times d}$ and $v \in \mathbb{R}^{n \times D}$, implementing the function

$$f(x) = \frac{1}{\sqrt{n}} v^\top \sigma(ux), \tag{2.20}$$

where σ denotes some activation function, and the input is $x \in \mathbb{R}^d$. Let $\mathcal{N}$ denote the initialization distribution over parameters u, v (usually Gaussian with proper scaling).

The Random Features or Gaussian Processes kernel

When u are fixed and only v are optimized, the NN represents a RF model with feature map $\psi(x) = \sigma(ux)$ and kernel

$$k(x, x') = \mathbb{E}_{u\sim\mathcal{N}}\left[\sigma(ux) \cdot \sigma(ux')\right]. \tag{2.21}$$

This correspondence extends to deep fully-connected NNs with more than two layers, where all weights before the output layer are fixed at their random initial values and only the output layer is trained. Kernel behavior is then obtained in the limit when the widths of the hidden layers go to infinity. With $f(x)$ denoting the output and v the output layer parameters of this DNN, the kernel definition (2.21) can be written more generally as

$$k(x, x') = \mathbb{E}_{v\sim\mathcal{N}}\left[\nabla_v f(x) \cdot \nabla_v f(x')\right]. \tag{2.22}$$

Note that because of its relevance in NNGPs, in current papers this kernel type is often referred to as the **GP kernel**.

Note that the RF or GP kernel describe the general kernel type, but the kernel resulting from computing (2.22) for a specific NN is architecture-dependent. In particular, different kernels can be expressed by varying the activation function $\sigma(\cdot)$. The popular ReLU activation $\sigma(x) = \max[0, x] = xH(x)$ and the Heaviside-step activation $\sigma(x) = H(x)$ correspond to the **arc-cosine kernel** of first and zeroth order, respectively.

The Arc-Cosine kernel

Cho and Saul (2009) aimed to combine the elegance of principled kernel learning with the expressive power of DL. For this purpose, they introduced a family of kernel functions called "arc-cosine" (AC) that would mimic NN computation in a kernel machine.

The lth-order AC kernel function is defined via the following integral representation:

$$K_l(x, y) = \frac{1}{(2\pi)^{d/2}} \int d^d w\, e^{-\|w\|^2/2} (w \cdot x)^l (w \cdot y)^l H(w \cdot x) H(w \cdot y) \tag{2.23}$$

for $x, y, w \in \mathbb{R}^d$, and with $H(\cdot)$ denoting the Heaviside step function. This integral can be solved analytically, and the final result can be expressed as

$$K_l(x, y) = \frac{1}{2\pi} \|x\|^l \|y\|^l J_l(\alpha_{x,y}), \tag{2.24}$$

where $\alpha_{x,y}$ is the angle between the inputs x and y,

$$\alpha = \arccos\left(\frac{x \cdot y}{\|x\|\|y\|}\right), \tag{2.25}$$

and $J_l(\alpha)$ is a family of functions defined in a closed-form expression provided in the original paper (Cho and Saul, 2009); the first few are:

$$J_0(\alpha) = \pi - \alpha\,, \tag{2.26}$$
$$J_1(\alpha) = \sin\alpha + (\pi - \alpha)\cos\alpha\,, \tag{2.27}$$
$$J_2(\alpha) = 3\sin\alpha\cos\alpha + (\pi - \alpha)(1 + 2\cos^2\alpha)\,. \tag{2.28}$$

The ACK functions arise in computation with the simplest kind of NN that consist of a single weight matrix W, and are equipped with a one-sided polynomial activation function $\sigma_l(z) = z^l H(z)$. Note that for $l = 0$, σ_0 is a step function – then the NN is an array of perceptrons, and for $l = 1$, σ_1 is the ReLU activation. The NN function maps inputs $x \in \mathbb{R}^d$ to outputs $f(x) \in \mathbb{R}^D$, with $f(x) = \sigma(Wx)$, where σ is applied element-wise, and the elements of W are randomly initialized from a Gaussian distribution, $W_{ij} \sim \mathcal{N}(0, 1)$. Consider now the inner product between two NN outputs:

$$f(x) \cdot f(y) = \sum_{i=1}^{D} (W_i \cdot x)^l H(W_i \cdot x)\,(W_i \cdot y)^l H(W_i \cdot y), \tag{2.29}$$

where W_i denotes the i-th row of the weight matrix W. In the limit where the NN width D goes to infinity, this inner product (normalized by D) yields the integral formula for the AC function:

$$\lim_{D\to\infty} \frac{1}{D} f(x) \cdot f(y) = K_l(x, y)\,, \tag{2.30}$$

Note that the order l of the ACK function corresponds to the order of NN activation function.

The ACK function will be relevant for the computations presented in the next chapter.

Neural Tangent kernel

In a standard setting, *all* layers of the NN are trained by stochastic gradient descent. In this case, it has been observed that, when the width of the hidden layers is very large, the model remains close to its *linearization* around its random initialization throughout

training, known as *"lazy training"* regime. Learning is then equivalent to a kernel method with another architecture-specific kernel, namely the **Neural Tangent Kernel** (NTK), popularized by Jacot et al. (2018). The NTK is defined as the inner product between the gradients of the NN outputs with respect to the NN parameters:

$$k(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \left[\nabla_\theta f(x) \cdot \nabla_\theta f(x') \right] . \tag{2.31}$$

2.5 Bayesian neural networks and Gaussian Processes

As mentioned in the beginning of the chapter, the correspondence between infinite NNs and GPs was fist studied in the context of Bayesian learning. The Bayesian approach to NNs is in general different from the NN-based ML methods currently prevalent in practice. In the study of NN kernels, the GP kernel plays a role in both the Bayesian and the non-Bayesian case. However, the distinction between these cases if often not stated clearly, which – I found – makes the topic of NN kernels confusing for non-experts. Therefore – even though my research presented in this thesis is not about Bayesian NNs – in this section I provide background information on the Bayesian perspective and GPs. Furthermore, because the original connection between NNs and kernels was made in the Bayesian framework, I find it interesting and valuable to retrace the line of thought which lead to these discoveries.

When NNs just came up in the 1990s, it was largely unclear how exactly to apply them to learning in practice, due to the lack of a principled approach to NN design, including the number of hidden units in a layer, the choice of activation function, the training hyper-parameters, etc. (Rasmussen and Williams, 2006). And while classical learning theory taught that one has to aim for the sweet spot in the bias-variance trade-off, to Bayesians it did not make sense to limit the number of parameters and make the model less capable.

In the Bayesian framework, which is radically different from conventional NN training, inference begins by specifying a *prior* distribution over model parameters. The prior encodes the user's assumptions (which are made based on knowledge, or educated guessing) about the relationships that are being modeled. By applying Bayes' theorem, the prior is then updated to the *posterior* based on observations (i.e., training data). Finally, predictions are made by averaging over all likely explanations under the posterior distribution.

Neal (1994) attempted to fuse NNs with Bayesian inference. The tricky part about this was identifying the proper prior: In NNs, the model parameters are the weights (and the biases), but since the weights do not have any direct interpretation, the *prior over weights* lacks any meaning. What matters is the *prior over functions* computed by the network, which is indirectly implied by the prior over the weights. Therefore, it is not immediately

clear how to design a prior distribution that expresses one's beliefs. Neal argued that since in the Bayesian framework the problem of overfitting does not occur, there is no reason to consider small NNs with limited expressivity, and thus proposed to use NNs with infinitely many hidden units. He established that in this limit, the prior over functions will converge to a Gaussian Process, making it possible to analyze the nature of the priors over functions that result from certain prior for the weights.

2.5.1 Gaussian Processes

Gaussian processes are stochastic processes. Intuitively, a stochastic *process* can be thought of as a generalization of a probability distribution: While a probability distribution describes a finite-dimensional random variable, a stochastic process describes *functions*. The Gaussian type is particularly favorable, as then the computations required for inference and learning become relatively easy.

Formally, a **stochastic process** (SP) is a collection of random variables $\{f(x) \mid x \in X\}$ indexed by a set X (Williams, 1998). In the supervised learning setting, X is $\mathbb{R}^d$, where d is the number of inputs. The SP is specified by the probability distribution for every finite subset of variables $f(x_1), \ldots, f(x_n)$ in a consistent manner. In the special case of a **Gaussian process** (GP), this distribution is a joint multivariate Gaussian:

$$p(f(x_1), \ldots, f(x_n)) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K}) \,. \tag{2.32}$$

The mean function $\boldsymbol{\mu} = (\mu(x_1), \ldots, \mu(x_n))$ and the covariance function $\boldsymbol{K}$ – also called the **kernel function** – with elements $K_{ij} = \operatorname{cov}(x_i, x_j)$ fully specify the GP:

$$\mu(x) = \mathbb{E}[f(x)] \tag{2.33}$$

$$\operatorname{cov}(x_i, x_j) = \mathbb{E}[(f(x_i) - \mu(x_i))\,(f(x_j) - \mu(x_j))] \tag{2.34}$$

The kernel function specifies all covariances in the model and thus encodes the prior of the GP. The prior is adjusted to a given application by choice of kernel.

Chapter 3

Are wider nets better given the same number of parameters?

The work presented in this chapter has been published as Golubeva et al. (2021).

In the previous chapter, I have provided a formal description of NN learning and introduced methods for a theoretical analysis of DL, in particular with a focus on illuminating the connection between NNs and kernel methods. In the current chapter, I present a study that applies some of the introduced theoretical methods to investigate the question posed in the title: Are wider nets better given the same number of parameters? This question is interesting from the theoretical perspective for any kind of NN and at the same time highly relevant for DL practice where very large models play the main role. As mentioned in chapter 1, overparametrization is a central characteristic property of DL models, and it is the property that puts DL into a conflict with statistical learning theory (SLT). According to SLT, overparametrized NN models are expected to generalize very poorly, but in practice we observe the exact opposite effect: DL models actually become better in their generalization ability when the number of parameters is increased.

By construction, the number of parameters in a NN is immediately related to the layer widths and the number of layers, i.e., the NN depth. While NN *depth* is the distinctive feature that marked the transition from the "old-school" *Machine Learning* to the practical and highly effective *Deep Learning*, it is constrained by the difficulties that it introduces into the training procedure. The main problem associated with training of deep NN models is the propagation of gradients during parameter updates, as required by the Gradient Descent algorithm. Larger depth makes gradients less stable, meaning that their magnitudes can fluctuate strongly, often leading to the problem known as *exploding or*

vanishing gradients. Therefore, the number of parameters in a deep NN is typically increased not by making it deeper, but by increasing its width. For instance, a popular DL model for image recognition called *ResNet* (which was introduced by He et al. (2015) and the name stands for "Residual Network") was improved upon by increasing layer width. Specifically, Zagoruyko and Komodakis (2017) have shown that a model now known as "WideResNet" with 16 layers outperforms all previous variants of ResNet models that all have more than 16 layers (standard ResNet variants have depths of 18, 34, 152 or 1001 layers).

The tradeoffs between depth and width for DL models have been studied quite extensively in a variety of context – see for instance Nguyen et al. (2021); Chatziafratis et al. (2019) and references therein. Similarly, a large number of works examined overparametrization of deep NNs, both in the context of optimization and generalization. However, in the vast majority of these works the number of parameters is increased by increasing NN width. In contrast, the study I present in this chapter aims at disentangling the effects of these very tightly connected quantities. The broader goal of this research work is to refine the notion of overparametrization, as understanding the effects and features of overparametrization is a step towards solving the puzzle of generalization in DL.

This analysis was conducted in a typical DL setting with a focus on the image classification task, using standard DL models and datasets. The target audience for this study is the ML community, and therefore some terms might sound unfamiliar to a physicist with little background in modern DL. However, we provide ample references for specific terms and an extensive appendix with details on every experiment. The research question that this study addresses is highly intuitive, and therefore easily accessible even for a ML novice, in particular given the introduction provided in the previous chapters and the paragraphs above.

3.1 Abstract

Empirical studies demonstrate that the performance of neural networks improves with increasing number of parameters. In most of these studies, the number of parameters is increased by increasing the network width. This begs the question: Is the observed improvement due to the larger number of parameters, or is it due to the larger width itself?

We compare different ways of increasing model width while keeping the number of parameters constant. We show that for models initialized with a random, static sparsity

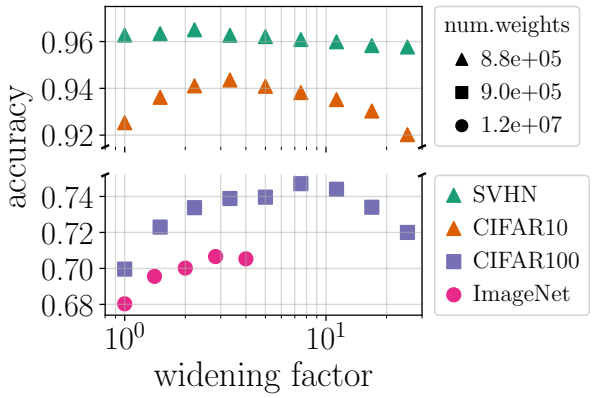


Figure 3.1: Test accuracy of ResNet-18 as a function of width. Performance improves as width is increased, even though **the number of weights is fixed**. Please see Section 3.4.3 for more details.

pattern in the weight tensors, network width is the determining factor for good performance, while the number of weights is secondary, as long as the model achieves high training accuracy. As a step towards understanding this effect, we analyze these models in the framework of Gaussian Process kernels. We find that the distance between the sparse finite-width model kernel and the infinite-width kernel at initialization is indicative of model performance.[1]

3.2 Introduction

Deep neural networks have shown great empirical success in solving a variety of tasks across different application domains. One of the prominent empirical observations about neural nets is that increasing the number of parameters leads to improved performance (Neyshabur et al., 2015, 2019; Hestness et al., 2017; Kaplan et al., 2020). This happens both for overparametrized networks, which have enough capacity to fit the training data, and in underparametrized settings. In the underparametrized regime, the improved performance can perhaps be attributed to the increase in the capacity due to the growing number of model parameters. However, the increased capacity by itself cannot explain the improved performance in the overparametrized setting, where the model already has enough capacity to fit the training data. This mystery has motivated many research directions fo-

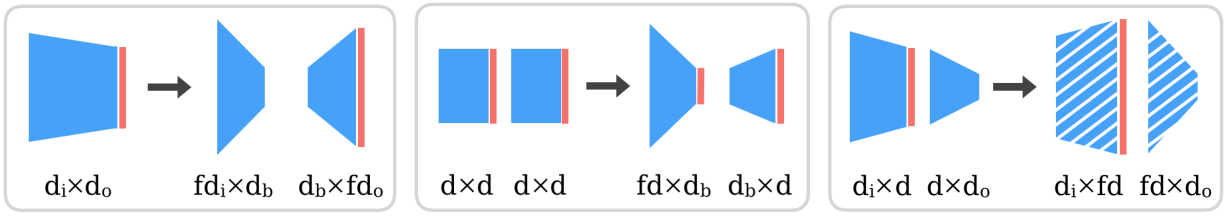
[1]Code is available at https://github.com/google-research/wide-sparse-nets

cused on this phenomenon, and the consequences of the effect for model optimization and generalization have been explored extensively. In the vast majority of these studies, both empirical and theoretical, the number of parameters is increased by increasing the width of the NN (Neyshabur et al., 2019; Du et al., 2019; Allen-Zhu et al., 2019). Network width itself on the other hand has been the subject of interest in studies analyzing its effect on the dynamics of NN optimization, e.g. using Neural Tangent Kernels (Jacot et al., 2018; Arora et al., 2019) and Gaussian Process Kernels (Wilson et al., 2016; Lee et al., 2018a).

All studies we know of suffer from the same fundamental issue: When increasing the width, the number of parameters is being increased as well, and therefore it is not possible to separate the effect of increasing width from the effect of increasing number of parameters. *How does each of these factors — width and number of parameters — contribute to the improvement in performance?* We conduct a principled study addressing this question, proposing and testing methods of increasing network width while keeping the number of parameters constant. Surprisingly, we find scenarios under which most of the performance benefits come from increasing the width.

Specifically, this work makes the following contributions:

- We propose three candidate methods (illustrated in Figure 3.2) for increasing network width while keeping the number of parameters constant.
  (a) **Linear bottleneck**: Substituting each weight matrix by a product of two weight matrices. This corresponds to limiting the rank of the weight matrix.
  (b) **Non-linear bottleneck**: Narrowing every other layer and widening the rest.
  (c) **Static sparsity**: Setting some weights to zero using a mask that is randomly chosen at initialization and remains static throughout training.
- **We show that performance can be improved by increasing the width, without increasing the number of model parameters.** We find that test accuracy can be improved using method (a) or (c), while method (b) only degrades performance. However, we find that (a) suffers from another degradation caused by the reparametrization, even before widening the network. Consequently, we focus on the sparsity method (c), as it leads to the best results and is applicable to any network type.
- We empirically investigate different ways in which random, static sparsity can be distributed among layers of the network and, based on our findings, propose an algorithm to do this effectively (Section 3.4.3).



(a) Linear Bottleneck (b) Non-linear Bottleneck (c) Static Sparsity

Figure 3.2: Schematic illustration of the methods we use to increase network width while keeping the number of weights constant. Blue polygons represent weight tensors, red stripes represent non-linear activations, and diagonal white stripes denote a sparsified weight tensor. We use f to denote the widening factor.

- We demonstrate that the improvement due to widening (while keeping the number of parameters fixed) holds across standard image datasets and models. Surprisingly, we observe that for ImageNet, increasing the width according to (c) leads to almost identical performance as when we allow the number of weights to increase along with the width (Section 3.4.3).
- To understand the observed effect theoretically, we study a simplified model and show that the improved performance of a wider, sparse network is correlated with a reduced distance between its Gaussian Process kernel and that of an infinitely wide network. We propose that reduced kernel distance may explain the observed effect (Section 3.5).

3.3 Related Work

Our work is similar in nature to the body of work studying the role of overparametrization and width. Neyshabur et al. (2015) observed that increasing the number of hidden units beyond what is necessary to fit the training data leads to improved test performance, and attributed this to the inductive bias of the optimization algorithm. Soltanolkotabi et al. (2018); Neyshabur et al. (2019); Allen-Zhu et al. (2019) further studied the role of over-parametrization in improving optimization and generalization. Woodworth et al. (2020) studied the implicit regularization of gradient descent in the over-parametrized setting, and Belkin et al. (2019) investigated the behavior at interpolation. Furthermore, Lee et al. (2018a) showed that networks at initialization become Gaussian Processes in the

large width limit, and Jacot et al. (2018) showed that infinitely wide networks behave as linear models when trained using gradient flow. Lee et al. (2020) systematically compared these different theoretical approaches. In all the above works, the number of parameters is increased by increasing the width. However, in this work, we conduct a controlled study of the effect of width by keeping the number of parameters fixed.

Perhaps Littwin et al. (2020) is the closest work to ours, which investigates over-parametrization achieved through ensembling as opposed to layer width. Ensembling is achieved by connecting several networks in parallel into one "collegial" ensemble. They show that for large ensembles the optimization dynamics simplify and resemble the dynamics of wide models, yet scale much more favorably in terms of number of parameters. However, the method employed there is borrowed from the ResNeXt architecture (Xie et al., 2016), which involves altering the overall structure of the network as width is increased. In this work we try to make minimal changes to network structure, in order to isolate the effect of width on network performance.

Finally, static sparsity is the basis of the main method we use to increase network width while keeping the number of parameters fixed. There is a large body of work on the topic of sparse neural networks (Park et al., 2016; Narang et al., 2017; Bellec et al., 2018; Frankle and Carbin, 2018; Elsen et al., 2019; Gale et al., 2019; You et al., 2019), and many studies derive sophisticated approaches to optimize the sparsity pattern (Lee et al., 2018b; Evci et al., 2019; Wang et al., 2020; Tanaka et al., 2020). In our study, however, sparsity itself is not the subject of interest. In order to minimize its effect in our controlled experiments, the sparsity pattern that we apply is randomly chosen and static. A recent work by Frankle et al. (2020) demonstrates that a random, fixed sparsity pattern leads to equal performance as the more involved methods for pruning applied at initialization, when the per-layer sparsity distribution is the same. However, we do not explore this direction in our study.

3.4 Empirical Investigation

In this section, we first explain our experimental methodology and then investigate the effectiveness of different approaches to increase width while keeping the number of parameters fixed. Finally, we discuss the respective roles of width and the number of parameters in improving performance.

3.4.1 Methodology

In order to identify how width and number of parameters separately affect performance, we need to decouple these quantities. For a fully-connected layer, width is the number of hidden units, and for a convolutional layer, width corresponds to the number of output channels. Increasing the width of one layer ordinarily entails an increase in the number of weights. Therefore, some adjustment is required to keep the total number of weights constant. This adjustment can be made at the level of layers by reducing some other dimension (i.e., by introducing a "bottleneck"), or at the level of weights by setting some of them to be zero. When choosing a method for our analysis, we take particular care about possible confounding variables, as many aspects of a neural network are intertwined. For example, we prefer to keep the number of non-linear layers in the network fixed, as it has been shown that changing it can significantly affect the expressive power, optimization dynamics and generalization properties of the network. With these constraints in mind, we specify three different methods listed above. In this section, we discuss these methods in detail, evaluate them, and present the results obtained on image classification tasks.

In summary, our approach is as follows: We select a network architecture which specifies layer types and arrangement (e.g., MLP with some number of hidden layers, or ResNet-18), set layer widths, and count the number of weights. We refer to this model as the *baseline*, and derive other models from it by increasing the width while keeping the number of weights constant. In this way, we obtain a family of models that we train using the same training procedure, and compare their test accuracies. For comparison and as a sanity check, we also consider the dense variants of the wider models.

We tested a variety of simple models, and observed the same general behavior in the context of our research question. For the discussion in this work, we focus on two model types: a MLP with one hidden layer, and a ResNet-18. We chose the ResNet-18 architecture (He et al., 2015) because it is a standard model widely used in practice. Its size is small enough that it allows us to increase the width substantially, yet it has sufficient representational power to obtain a nontrivial accuracy on standard image datasets. When creating a family of ResNet-18 models, we refer to the number of output channels of the first convolutional layer as *the width*, and do not alter the default width ratios of all following layers. Further details of each experiment and figure are specified in Appendix A.

3.4.2 Bottleneck methods

Here we discuss the two bottleneck methods described in the previous section and in Figure 3.2.

Linear Bottleneck

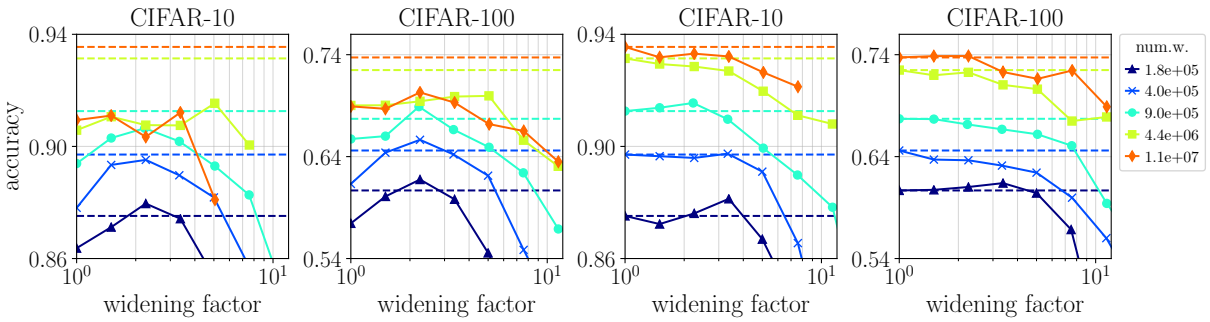
Substitute each weight matrix $W \in \mathbb{R}^{d_i \times d_o}$ by $W_1 W_2$, where $W_1 \in \mathbb{R}^{d_i \times d_b}$, $W_2 \in \mathbb{R}^{d_b \times d_o}$ and $d_b \leq \min(d_i, d_o)$. In the case of a ResNet-18 model, each convolutional layer is replaced by two convolutional layers with kernel dimensions 3×3 and 1×1, respectively, and with $d_b = d_i = d_o$ before widening. Note that if $d_b = \min(d_i, d_o)$, then this reparametrization has the exact same expressive power[2] as the original network. Now, one can make the network wider by increasing the number of hidden units (d_i and d_o here) and reducing the bottleneck size d_b. This would correspond to imposing a low-rank constraint on the original weight matrix W. Linear bottleneck has been proposed as a way to reduce the number of parameters and improve training speed (Ba and Caruana, 2014; Urban et al., 2016). The caveat of this method is that, even though the expressive power of the reparametrized network is the same as the original one, the reparametrization changes the gradient descent trajectory and hence can affect the final results. It is therefore not possible to control the implicit regularization caused by this transformation. Note that substituting the weight matrix by a product of two matrices changes the number of parameters of the model.

Non-linear Bottleneck

One way to create a non-linear bottleneck is to split each layer in two as described above and add a non-linearity to the first layer. The problem with this approach is that adding a non-linearity changes the expressive power of the model, and hence adds a factor that we cannot control. An alternative approach, which is more favorable in our case and which we adopt here, is to modify the layers in pairs. The input dimension of the first layer is increased, while its output dimension (and consequently the input dimension of the second layer) is reduced ($d_b < d$). Non-linear bottlenecks are widely used in practice, particularly as a way to save parameters in very deep architectures, such as deeper variants of ResNet (He et al., 2016) and DenseNet (Huang et al., 2017).

The performance of these methods on CIFAR-10 and CIFAR-100 datasets is demonstrated in Figure 3.3. The results indicate that increasing width using the linear bottleneck indeed leads to improved accuracy up to a certain width. Moreover, this effect is more pronounced in smaller models that are less overparametrized. This is similar to the improvement gained when increasing the width along with the number of parameters (without a bottleneck): The improvement tends to be diminished when the base network is wider. However, as discussed above, the act of substituting a weight matrix by a product of two

[2]Here, expressive power refers to the set of functions that can be represented by the network.



(a) Linear bottleneck (b) Non-linear bottleneck

Figure 3.3: Best test accuracy obtained by ResNet-18 models widened using the bottleneck methods. The performance of the baseline network (before introducing bottleneck layers) is denoted by dashed lines. The legend indicates the number of weights in the baseline network. In (a), it is equal to the number of weights before introducing bottleneck layers. In (b), it is equal to the total number of weights.

weight matrices changes the optimization trajectory which could in turn affect generalization. Indeed, the performance of the original model, indicated by dashed horizontal lines in the Figure, is significantly higher than that of the transformed models. Therefore, even though the width increase at a constant number of weights with the linear bottleneck method improves the result of the most narrow model obtained after the transformation, it typically does not outperform the default ResNet-18 model. Due to lack of control over the effect of inductive bias, we conclude that this choice does not qualify to be used in our controlled experiments.

The non-linear bottleneck method does not suffer from the same issues as the linear version in terms of inductive bias of reparametrization. However, as Figure 3.3 demonstrates, no empirical improvement is obtained by increasing the width, except for the model with $1.8e+05$ weights (and then the improvement is mild). Therefore, we conclude that the non-linear bottleneck model does not show a significant enough improvement to be considered as an effective method for our study.

3.4.3 Sparsity method

We now turn to the sparsity method illustrated in Figure 3.2c, which is the main method studied in this work. We start with a baseline model that has dense weight tensors. We then increase the width by a widening factor f and sparsify the weight tensors such that

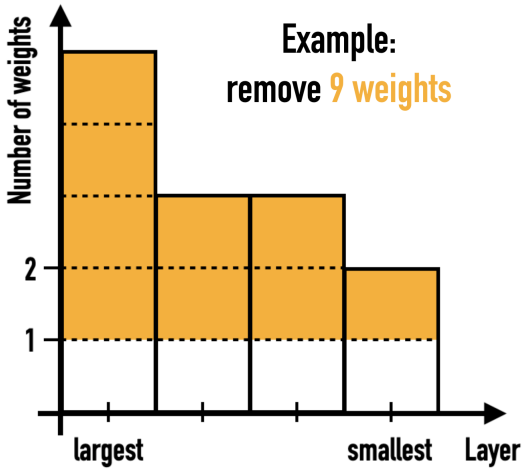
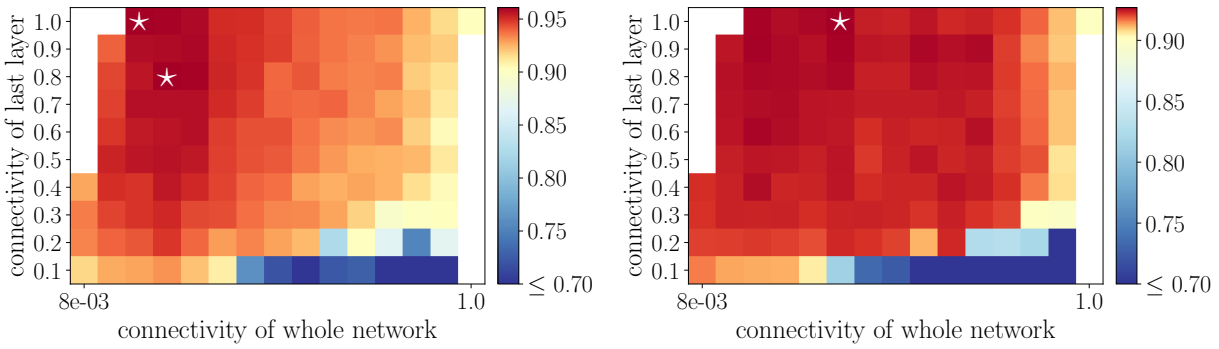


Figure 3.4: Illustration of our algorithm for distributing sparsity over model layers on a toy example: a NN with four layers and a total of 13 weights. The goal is to remove nine weights. Orange color indicates how these nine are distributed over the NN layers. The code implementing this algorithm is included in Appendix D.

the total number of trainable weights is the same as in the baseline model. In an attempt to run a controlled experiment and minimize the differences between the sparsified setup and the baseline setup, we choose the sparsity mask at random at initialization and keep it static during training. That is, the weights to remove are chosen at random, and their values are set to zero throughout training. In this sense, our method differs from most other pruning methods discussed in the literature, where the aim is to maximize performance through optimized sparsity. The advantage of the sparsity method over the bottleneck methods considered earlier is that it allows us to control the number of weights without altering the overall network structure. We define the *connectivity* of a sparse model to be the ratio between the number of its parameters and the number of parameters in a dense model of the same width.

In order to implement the sparsity method, we need to choose how to distribute the sparsity both across network layers and within each layer. To prevent smaller layers from being cut out entirely, we choose the number of weights to be removed in each layer to be proportional to the number of weights in that layer (except for BatchNorm layers, which are kept intact). Figure 3.4 demonstrates the principle of our algorithm for sparsity distribution. Within each layer, we distribute the sparsity uniformly across all dimensions of the weight tensor. Overall, this method of distributing the weights led to the best



(a) ReLU (b) Linear

Figure 3.5: Test accuracy (color-coded) on MNIST obtained by MLP models with 1 hidden layer and ReLU or linear activations, as a function of overall connectivity (horizontal axis) and of connectivity in the last layer (vertical axis). White stars indicate points with the highest test accuracy.

performance among the different methods we tried in our experiments (see Appendix E for more details).

MLP

We study a fully-connected network with one hidden layer trained on MNIST. We use this simple setup to compare different sparsity distribution patterns across layers for a given network connectivity, and to compare the effects of increased width with and without ReLU non-linearities. Specifically, we consider a linear model to address the fact that width increase introduces additional ReLU units and thus potentially enables the network to express a more complex function.

The dense MLP at width n has two weight matrices: The first layer matrix has size $784 \times n$ and the last layer has size $n \times 10$. In the experiments, we set the total number of weights to 3970 and consider widths ranging from 5 to 640, corresponding to network connectivities between 1 and $5/640 \approx 0.008$. At each width, we vary the connectivity of the last layer (the smaller of the two) between 1.0 and 0.1. Given a fixed total number of weights, each valid combination of width and last-layer connectivity determines a model (up to the random sparsity pattern, which depends only on the random seed).

For both ReLU and Linear activation, sparse wide models can outperform the dense baseline, as shown in Figure 3.5. We find that sparse, wide models can outperform the

dense baseline models for both ReLU and linear activations. The ReLU model attains its maximum performance at around $3-6\%$ connectivity, which corresponds to a widening factor of 16 or 32. At the optimal point the connectivity of the last layer is high, above 80%. It is therefore advantageous in this case to remove more weights from the first layer than from the last layer. This makes intuitive sense: Removing weights from a layer that starts out with fewer weights can be expected to make optimization more difficult. This result motivated our choice to remove weights proportionally to layer size when sparsifying other models. Finally, the fact that larger width leads to improvement even in the deep linear model implies that the improvement cannot be attributed solely to the increased model capacity due to additional non-linearities in the network function.

ResNet-18

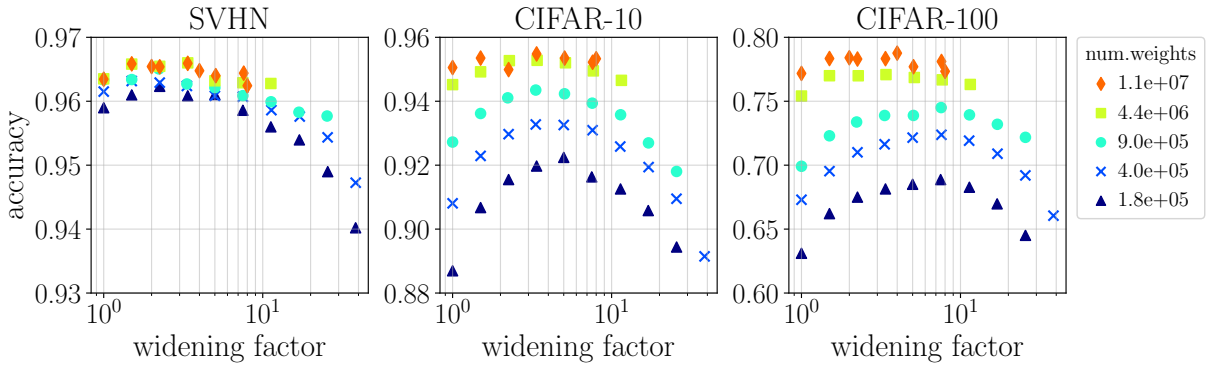
We train families of ResNet-18 models on ImageNet, CIFAR-10, CIFAR-100 and SVHN, covering a range of widths and model sizes. A detailed example of the sparsity distribution over all layers is shown in Appendix F.

Table 3.1 shows the results obtained on ImageNet. As expected, the performance improves as the width and the number of weights increase (row 1). However, up to a certain width, a comparable improvement is achieved when only the width grows and the number of weights remains fixed (row 2). We believe the reason for declining test accuracy at yet larger widths is that the connectivity becomes too small and impairs the trainability of the model, as we observe that the training accuracy deteriorates along with the test accuracy. Therefore, in this case the determining factor for model performance is width rather than number of parameters. Figure 3.6 shows the results of training the ResNet-18 model families on several additional datasets. We again find that performance improves with width at a fixed number of parameters, up to a certain widening factor. The effect is

width	64	90	128	181	256
dense	68.03 (11.7)	69.11 (22.8)	70.22 (45.7)	70.91 (90.7)	71.89 (180.6)
sparse	–	69.56 (11.7)	70.02 (11.7)	70.66 (11.7)	70.53 (11.7)

Table 3.1: ResNet-18 on ImageNet: Top-1 test accuracy (in %) and in parentheses the number of weights in millions. All sparse models have the same number of weights as the smallest dense model, yet the improvement obtained with increasing width is on par with the dense models, up to a certain width.

(a) Test accuracy



(b) Training accuracy

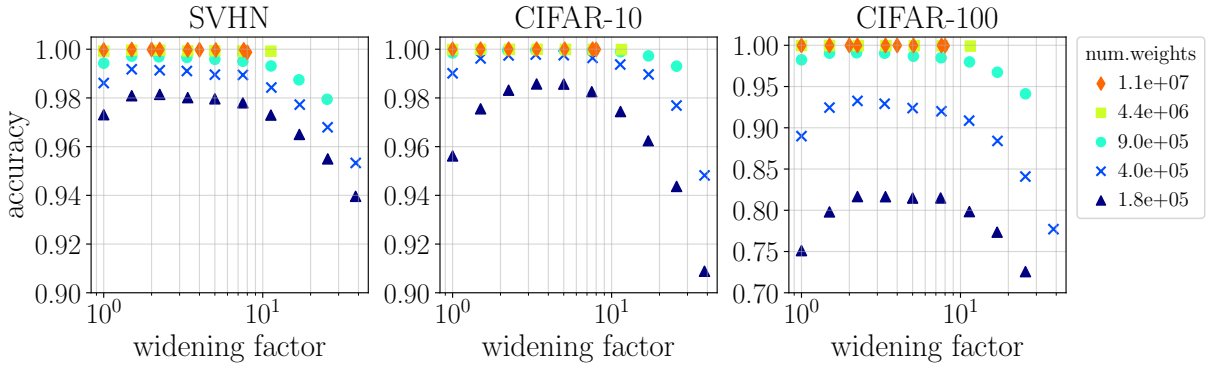


Figure 3.6: Best **test** accuracy (a) and corresponding **training** accuracy (b) obtained by ResNet-18 models of different width and size (approximate total number of weights is shown in the legend). The leftmost data point of each color corresponds to the dense baseline model, and all subsequent data points correspond to its wider and sparser variants. The decline of performance at larger widening is because sparsity at these levels harms the optimization procedure, making the training accuracy deteriorate. See Appendix B for the same data plotted as a function of network connectivity, and for the version with error bars.

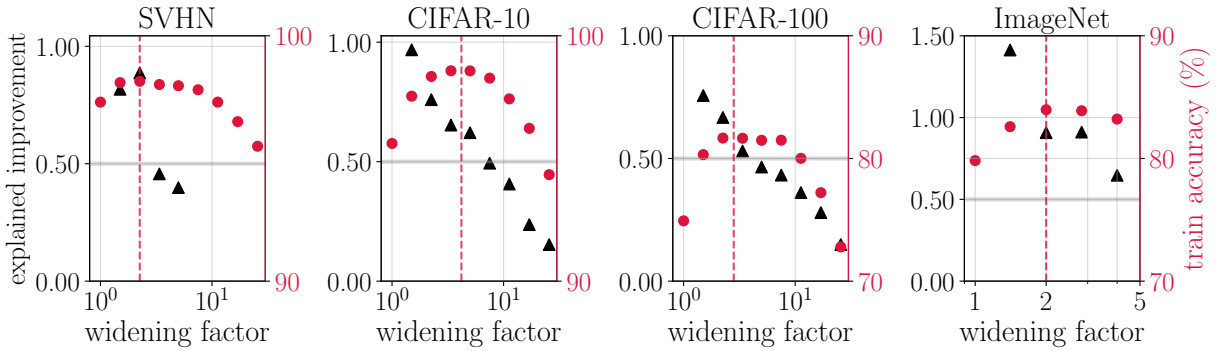


Figure 3.7: The fraction of test accuracy improvement (black triangles) obtained by widening the model without increasing the number of weights, compared to increasing the number of weights along with the width. The dashed vertical line indicates widening factor for maximal training accuracy (red circles). Note that for ImageNet, at width 90 (widening factor 1.4) the sparse model attained a higher test accuracy than the dense model, as reported in Table 3.1.

most pronounced for more difficult tasks and for smaller models that do not reach 100% training accuracy, yet it is still present for models that do fit the training set.

Figure 3.7 compares the performance improvement of a sparse, wide model against that of a dense model with the same width. In particular, it shows the fraction of the improvement that can be attributed to width alone – that is, the ratio between the sparse/baseline accuracy gap and the dense/baseline accuracy gap. We see that as long as the model can achieve high training accuracy, most of the improvement in performance can be attributed to the width.

3.5 Theoretical analysis in a simplified setting

We showed empirically that wide, sparse NNs with random connectivity patterns can outperform dense, narrower NNs when the number of parameters is fixed. It is well known that wider (dense) NNs can achieve consistently better performance. When the NN is sparse, its performance as a function of width has a maximum. In this section we investigate a potential theoretical explanation for this effect building on the connection between NNs and kernel learning discussed in the previous chapter.

Specifically, we conjecture that the kernel of a finite-width NN at initialization is in-

dicative of its performance, and that optimal performance is achieved when its distance to the infinite-width kernel is minimized. We further hypothesize that this distance can be reduced by increasing the NN width at a fixed number of parameters. In the following, we formalize this conjecture and derive expressions for the kernel of a sparse finite-width NN with one hidden layer. We calculate the kernel distance theoretically, and show that the distance predicted using this result is in good agreement with experiments.

Consider a 2-layer ReLU network with the function $f : \mathbb{R}^d \to \mathbb{R}$ given by

$$f(x) = \frac{1}{\sqrt{nd}}\, v^T \left[ux\right]_+ , \qquad [z]_+ := zH(z)\,. \tag{3.1}$$

Here, $x \in \mathbb{R}^d$ is the input, the network parameters are $u \in \mathbb{R}^{n\times d}$ and $v \in \mathbb{R}^{n\times D}$, and $H(\cdot)$ is the Heaviside step function. For simplicity, we omit the biases, and set the output dimension $D = 1$. Note that in this section we use *NTK parametrization*, whereas in previous sections we used LeCun parametrization (see Appendix C). Each network parameter is sampled independently from $\mathcal{N}(0, \sigma^2)$ with probability p, and is set to zero with probability $1 - p$. We use the variable $\omega \in \mathbb{R}$ representatively for all network parameters u, v. The probability density function for each parameter ω is therefore given by

$$\Pr(\omega) = \frac{p}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\omega^2}{2\sigma^2}\right) + (1-p)\delta(\omega)\,. \tag{3.2}$$

where $\delta(\cdot)$ is the Dirac delta function. We set $\sigma^2 = p^{-1}$.

In the following, we consider the *Gaussian Process* (GP) kernel of the network, defined as

$$\Theta_{\mathrm{GP}}(x, y) := \nabla_v f(x)^\top \nabla_v f(y)\,, \tag{3.3}$$

where the gradients are taken with respect to elements of the last layer only. Note that

$$\partial_{v_i} f(x) = \frac{1}{\sqrt{nd}}\, \partial_{v_i} \sum_j v_j \left([ux]_+\right)_j = \frac{1}{\sqrt{nd}} \left([ux]_+\right)_i \tag{3.4}$$

and thus

$$\nabla_v f(x) = \left(\partial_{v_1} f(x), \ldots, \partial_{v_n} f(x)\right)^\top = \frac{1}{\sqrt{nd}} \left[ux\right]_+ , \tag{3.5}$$

such that the inner product in the definition of the GP kernel is essentially the inner product of the *"hidden feature vectors"*, i.e., outputs of the hidden layer:

$$\Theta_{\mathrm{GP}}(x, y) = \nabla_v f(x)^\top \nabla_v f(y) = \frac{1}{nd} \left[ux\right]_+^\top \left[uy\right]_+ . \tag{3.6}$$

Therefore, we can use the *arc-cosine kernel* (ACK) functions introduced in section 2.4 to express $\Theta_{\rm GP}(x,y)$ for a dense NN. Let $\Theta^n_{\rm GP}$ be the *dense* kernel (with $p=1$) at width n, and let $\Theta^\infty_{\rm GP} = \lim_{n\to\infty}\Theta^n_{\rm GP}$ be the dense infinite-width kernel. Because in the infinite-width limit the kernel is given *exactly* by the relation (2.30), its mean and variance are straightforwardly obtained:

$$\mathbb{E}_\omega[\Theta^\infty_{\rm GP}(x,y)] = \frac{1}{d}\,K_1(x,y)\,, \tag{3.7}$$

$$\mathrm{Var}[\Theta^\infty_{\rm GP}(x,y)] = 0\,, \tag{3.8}$$

with $K_1(x,y)$ given by (2.23).

To account for the case of a sparse NN with connectivity p, we generalize the definition (2.23) for the ACK function of l-th order as follows:

$$\tilde{K}_{l,p}(x,y) := \int d^d w \prod_{i=1}^d \Pr(w_i)(w\cdot x)^l (w\cdot y)^l H(w\cdot x)H(w\cdot y) \tag{3.9}$$

for $x,y,w\in\mathbb{R}^d$, and with the random variables w_i distributed according to (3.2). Note that for $p=1$ we recover the original integral (2.23), i.e., $\tilde{K}_{l,1}(x,y) = K_l(x,y)$.

We can express $\tilde{K}_{l,p}(x,y)$ in terms of $K_l(x,y)$ by doing the following transformation: We introduce a vector $s=(s_1,\ldots,s_d)$ with elements in $\{0,1\}$ which acts as a mask for the vectors w,x,y, and denote the masked vectors by the subscript $_s$, with $w_s = (w_1 s_1,\ldots,w_d s_d)$ and analogously for x_s, y_s. Note that the elements of s are random variables drawn from a binomial distribution with probability p for value 1, and the 2-norm of s, denoted $\|s\|$, is equal to the number of nonzero elements in s. Using the mask s allows us to restrict the integration to non-zero elements of w, namely $d^{\|s\|}w_s = \prod_{i|s_i=1} dw_i$:

$$\begin{aligned}
\tilde{K}_{l,p}(x,y) &= (2\pi\sigma^2)^{-\|s\|/2} \sum_{s_1,\ldots,s_d\in\{0,1\}} p^{\|s\|}(1-p)^{d-\|s\|} \\
&\qquad \times \int d^{\|s\|}w_s\, e^{-\|w_s\|^2/2\sigma^2} (w_s\cdot x)^l (w_s\cdot y)^l H(w_s\cdot x)H(w_s\cdot y) \\
&= \sigma^{2l}\cdot \sum_{s_1,\ldots,s_d\in\{0,1\}} p^{\|s\|}(1-p)^{d-\|s\|}\; K_l(x_s,y_s)\,.
\end{aligned} \tag{3.10}$$

Consider now the *sparse* kernel $\Theta_{\rm GP}$. We want to compute its mean over all parameter

initializations (i.e., the ensemble average) and its variance, that is,

$$\mathbb{E}_\omega[\Theta_{\rm GP}] = \int_{-\infty}^{\infty} d^L\omega \prod_{i=1}^{L} \Pr(\omega_i)\, \Theta_{\rm GP}\,, \tag{3.11}$$

$$\mathrm{Var}[\Theta_{\rm GP}] = \mathbb{E}_\omega\big[\Theta_{\rm GP}^2\big] - \mathbb{E}_\omega[\Theta_{\rm GP}]^2\,. \tag{3.12}$$

The mean is easily obtained:

$$\mathbb{E}_\omega[\Theta_{\rm GP}(x,y)] = \mathbb{E}_\omega\left[\frac{1}{nd}\,[ux]_+^\top [uy]_+\right] \tag{3.13}$$

$$= \frac{1}{nd}\,\mathbb{E}_\omega\left[\sum_{k=1}^{n}(u_k\cdot x)H(u_k\cdot x)\;(u_k\cdot y)H(u_k\cdot y)\right]\,, \tag{3.14}$$

where we have used u_k to denote the k-th row of $u\in\mathbb{R}^{n\times d}$. The contribution is the same for each of the n rows. Changing to integral form of the expectation value we obtain:

$$\begin{aligned}\mathbb{E}_\omega[\Theta_{\rm GP}(x,y)] &= \frac{1}{d}\int_{-\infty}^{\infty} d^d u_k \prod_{i=1}^{d}\Pr(u_{k_i})\,(u_k\cdot x)H(u_k\cdot x)\,(u_k\cdot y)H(u_k\cdot y)\\ &= \frac{1}{d}\,\tilde{K}_{1,p}(x,y)\,.\end{aligned} \tag{3.15}$$

To obtain the variance, we need to compute $\mathbb{E}_\omega[\Theta_{\rm GP}(x,y)^2]$:

$$\begin{aligned}\mathbb{E}_\omega\big[\Theta_{\rm GP}(x,y)^2\big] = \frac{1}{n^2d^2}\\ \times\sum_{k,k'=1}^{n}\mathbb{E}_\omega[(u_k\cdot x)H(u_k\cdot x)(u_k\cdot y)H(u_k\cdot y)(u_{k'}\cdot x)H(u_{k'}\cdot x)(u_{k'}\cdot y)H(u_{k'}\cdot y)]\end{aligned}$$

For $k=k'$, we have n equal terms:

$$\begin{aligned}I_{k=k'} &= \sum_{k=1}^{n}\mathbb{E}_\omega\big[(u_k\cdot x)^2H(u_k\cdot x)(u_k\cdot y)^2H(u_k\cdot y)\big]\\ &= n\int_{-\infty}^{\infty} d^d\omega\prod_{i=1}^{d}\Pr(\omega_i)(\omega\cdot x)^2H(\omega\cdot x)(\omega\cdot y)^2H(\omega\cdot y)\\ &= n\,\tilde{K}_{2,p}(x,y)\,.\end{aligned} \tag{3.16}$$

For $k \neq k'$, we obtain a contribution of $n(n-1)$ equal terms:

$$\begin{aligned} I_{k\neq k'} &= \sum_{k=1}^{n} \sum_{k\neq k} \mathbb{E}_\omega[(u_k \cdot x)H(u_k \cdot x)(u_k \cdot y)H(u_k \cdot y)(u_{k'} \cdot x)H(u_{k'} \cdot x)(u_{k'} \cdot y)H(u_{k'} \cdot y)] \\ &= n(n-1) \int d^d u_k \prod_{i=1}^{d} \Pr(u_{k_i})(u_k \cdot x)H(u_k \cdot x)(u_k \cdot y)H(u_k \cdot y) \\ &\qquad \times \int d^d u_{k'} \prod_{i=1}^{d} \Pr\big(u_{k'_i}\big)(u_{k'} \cdot x)H(u_{k'} \cdot x)(u_{k'} \cdot y)H(u_{k'} \cdot y) \\ &= n(n-1)\, \tilde{K}_{1,p}(x,y)^2 \,. \end{aligned} \tag{3.17}$$

Putting them together, we obtain:

$$\mathbb{E}_\omega\big[\Theta_{\mathrm{GP}}(x,y)^2\big] = \frac{1}{n^2 d^2}\left[n\tilde{K}_{2,p}(x,y) + n(n-1)\tilde{K}_{1,p}(x,y)^2\right] \tag{3.18}$$

In summary, the mean and variance for the sparse Θ_{GP} are:

$$\mathbb{E}_\omega[\Theta_{\mathrm{GP}}(x,y)] = \frac{1}{d}\,\tilde{K}_{1,p}(x,y)\,, \tag{3.19}$$

$$\begin{aligned} \mathrm{Var}[\Theta_{\mathrm{GP}}(x,y)] &= \frac{1}{n^2 d^2}\left[n\tilde{K}_{2,p}(x,y) + n(n-1)\tilde{K}_{1,p}(x,y)^2\right] - \frac{1}{d^2}\tilde{K}_{1,p}(x,y)^2 \\ &= \frac{1}{nd^2}\left[\tilde{K}_{2,p}(x,y) - \tilde{K}_{1,p}(x,y)^2\right]. \end{aligned} \tag{3.20}$$

The *mean squared distance* between the sparse and the infinite-width GP kernel is thus:

$$\begin{aligned} \mathbb{E}_\omega\big[(\Theta_{\mathrm{GP}}(x,y) - \Theta^{\infty}_{\mathrm{GP}}(x,y))^2\big] &= [\mathbb{E}_\omega[\Theta_{\mathrm{GP}}(x,y)] - \Theta^{\infty}_{\mathrm{GP}}(x,y)]^2 + \mathrm{Var}[\Theta_{\mathrm{GP}}(x,y)] \\ &= \frac{1}{d^2}\left[\tilde{K}_{1,p}(x,y) - K_1(x,y)\right]^2 + \frac{1}{d^2 n}\left[\tilde{K}_{2,p}(x,y) - \tilde{K}_{1,p}(x,y)^2\right]. \end{aligned} \tag{3.21}$$

Next, we derive an approximate form of this expression (3.21) using plausible arguments.

3.5.1 Approximating the kernel distance

The following derivation is not rigorous, but we compare the results against an empirical calculation (see Figure 3.8) and find good agreement. In this derivation, we make two

assumptions: (1) We assume $dp \gg 1$, which is easily fulfilled in practice, as the input dimension d is ~ 1000 even for the smaller image datasets (e.g., $d = 784$ for MNIST). (2) We assume that the inputs $x, y \in \mathbb{R}^d$ are independent random vectors with elements sampled from $\mathcal{N}(0, 1)$. This is intuitively justified by the fact that normalizing input data to have zero mean and unit variance is standard in ML.

For given p, we expect the dominant contribution to $\tilde{K}_{l,p}$ in the sum (3.10) to come from terms where $\|s\| = \sum_i s_i \approx dp$, and so we consider a mask s with this property. x_s and y_s are then random vectors with effective dimension dp, and we can approximate $x_s \cdot y_s \approx dp$ and $\|x_s\| \approx \sqrt{dp}(1 - 1/4dp)$ (and similarly for $\|y_s\|$). Here we used the relation for random vectors of dimension d: $\mathbb{E}\left[\|x\|^l\right] = 2^{l/2}\frac{\Gamma((d+l)/2)}{\Gamma(d/2)}$. We denote the angle between x, y by α, and the angle between x_s, y_s by α_s. Then, from the above we expect:

$$\cos\alpha_s \approx \frac{\xi}{\sqrt{dp}}\,, \quad \sin\alpha_s \approx \sqrt{1 - \frac{1}{dp}} \approx 1 - \frac{1}{2dp}\,, \quad \alpha_s \approx \frac{\pi}{2} - \frac{\xi}{\sqrt{dp}}\,, \tag{3.22}$$

where $\xi = \pm 1$ is a random sign.

Next, we consider the integrals $K_l(x_s, y_s)$, specifically their analytical solution (2.24). Using the random vector approximations, we find

$$K_1(x_s, y_s) = \frac{1}{2\pi}\|x_s\|\|y_s\|J_1(\alpha_s) \approx \frac{dp}{2\pi}\left[1 + \frac{\pi\xi}{2\sqrt{dp}}\right]\,, \tag{3.23}$$

$$K_2(x_s, y_s) = \frac{1}{2\pi}\|x_s\|^2\|y_s\|^2 J_2(\alpha_s) \approx \frac{(dp)^2}{2\pi}\left[\frac{\pi}{2} + \frac{4\xi}{\sqrt{dp}} + \frac{\pi}{dp} + \frac{\xi}{2(dp)^{3/2}}\right]\,. \tag{3.24}$$

For the sparse functions $\tilde{K}_l(x, y)$, we assume as before that the contribution from the sum over masks is concentrated where the mask $\Sigma_i s_i \approx dp$. For a mask s obeying this condition we then have $\tilde{K}_{l,p}(x, y) \approx \sigma^{2l} K_l(x_s, y_s)$, and specifically with $\sigma^2 = p^{-1}$:

$$\tilde{K}_{1,p}(x, y) \approx \frac{d}{2\pi}\left[1 + \frac{\pi\xi}{2\sqrt{dp}} + \frac{1}{2dp}\right]\,, \tag{3.25}$$

$$\tilde{K}_{2,p}(x, y) \approx \frac{d^2}{2\pi}\left[\frac{\pi}{2} + \frac{4\xi}{\sqrt{dp}} + \frac{\pi}{dp} + \frac{\xi}{2(dp)^{3/2}}\right]\,. \tag{3.26}$$

The diagonal elements where $x = y$ and $\alpha_s = \alpha = 0$ are given by $\tilde{K}_{l,p}(x, x) \approx \sigma^{2l} K_l(x_s, x_s) \approx \frac{\sigma^{2l}(dp)^l}{2\pi}\left[1 + \frac{l(l-1)}{dp}\right] J_l(0)$. In particular,

$$\tilde{K}_{1,p}(x, x) \approx \frac{d}{2}\,, \tag{3.27}$$

$$\tilde{K}_{2,p}(x, x) \approx \frac{3d^2}{2}\left(1 + \frac{2}{dp}\right)\,. \tag{3.28}$$

Finally, using the approximations (3.25), (3.26), (3.27), (3.28), we obtain the approximate formula for the squared kernel distance (3.21):

$$\mathbb{E}_\omega\left[(\Theta_{\text{GP}}(x,y)-\Theta^\infty_{\text{GP}}(x,y))^2\right] \approx \frac{1}{4d}\left[\frac{1}{4}\left(\frac{1}{\sqrt{p}}-1\right)^2+\frac{d}{n}\left(1-\frac{1}{\pi^2}\right)\right], \tag{3.29}$$

$$\mathbb{E}_\omega\left[(\Theta_{\text{GP}}(x,x)-\Theta^\infty_{\text{GP}}(x,x))^2\right] \approx \frac{1}{n}\left(\frac{5}{4}+\frac{3}{dp}\right). \tag{3.30}$$

We would like to find the minimum of the distance (3.29) when the mean number of parameters np is fixed. Roughly, we would like to minimize $\frac{1}{4}(p^{-1/2}-1)^2+\frac{dp}{np}$ with np constant. Assuming that the minimum is at $\sqrt{p}\ll 1$, we find that the minimum is at

$$p_* \approx \sqrt{\frac{np}{4d}}. \tag{3.31}$$

3.5.2 Summary of results

We consider the GP kernel Θ_{GP} of a 2-layer network with ReLU activations, input $x\in\mathbb{R}^d$, hidden layer width n, connectivity p, and with the weights sampled from $\Pr(\omega)$.

The mean squared distance between Θ_{GP} and the (dense) infinite-width kernel $\Theta^\infty_{\text{GP}}$ is

$$\begin{aligned}\mathbb{E}_\omega\left[(\Theta_{\text{GP}}(x,y)-\Theta^\infty_{\text{GP}}(x,y))^2\right] &= \frac{1}{d^2}\left[\tilde{K}_{1,p}(x,y)-K_1(x,y)\right]^2 \\ &\quad+\frac{1}{d^2 n}\left[\tilde{K}_{2,p}(x,y)-\tilde{K}_{1,p}(x,y)^2\right].\end{aligned} \tag{3.32}$$

Here,

$$\tilde{K}_{l,p}(x,y)=\sigma^{2l}\cdot\sum_{s_1,\ldots,s_d\in\{0,1\}} p^{\sum_i^d s_i}(1-p)^{d-\sum_i^d s_i}K_l(x_s,y_s), \tag{3.33}$$

$$K_l(x,y)=\frac{1}{(2\pi)^{d/2}}\int_{-\infty}^{\infty} d^d w\, e^{-\|w\|^2/2}(w\cdot x)^l(w\cdot y)^l H(w\cdot x)H(w\cdot y), \tag{3.34}$$

and $s=(s_1,\ldots,s_d)$ is a vector with elements in $\{0,1\}$, and $x_s:=(s_1x_1,\ldots,x_ds_d)$ is the vector x with some elements zeroed out.

The following closed-form approximation to the kernel distance holds under the assumptions that the input vectors x,y are independent random vectors and $pd\gg 1$:

$$\mathbb{E}_\omega\left[(\Theta_{\text{GP}}(x,y)-\Theta^\infty_{\text{GP}}(x,y))^2\right] \approx \frac{1}{4d}\left[\frac{1}{4}\left(\frac{1}{\sqrt{p}}-1\right)^2+\frac{d}{n}\left(1-\frac{1}{\pi^2}\right)\right]. \tag{3.35}$$

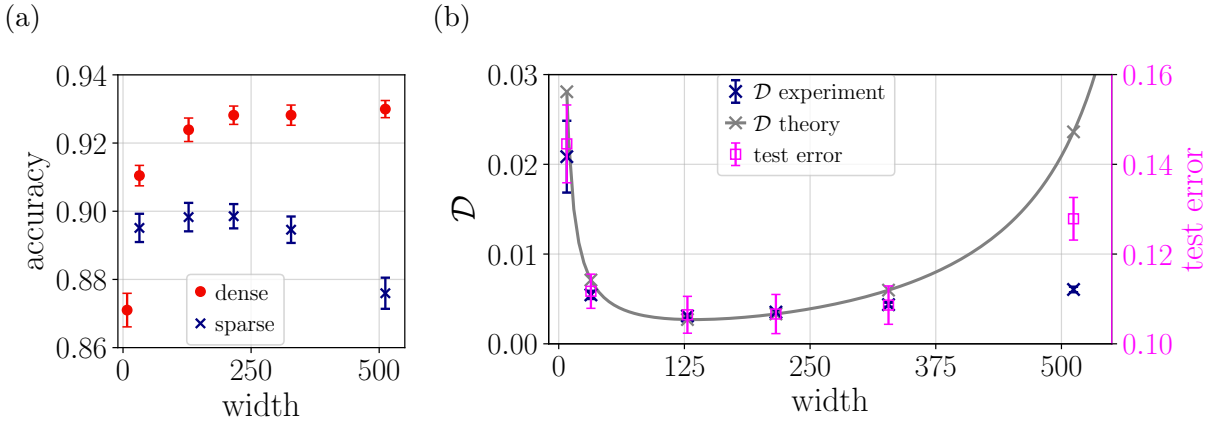


Figure 3.8: MLP with one hidden layer and no biases trained on a subset of MNIST. (a) Test accuracy achieved by dense (filled circles) and sparse (crosses) models of different width. (b) Mean squared distance $\mathcal{D}$ of the sparse model's kernel from the infinite-width model's kernel computed at initialization (both experimental (blue) and theoretical (gray) result), and the test error attained by trained models (pink). The empirical distance $\mathcal{D}$ is obtained by averaging the squared distance $(\Theta_{\text{GP}}(x, x') - \Theta^{\infty}_{\text{GP}}(x, x'))^2$ over 10^4 pairs of test samples and over 10 random initializations. See Appendix A for additional details.

In order to keep the number of parameters fixed as we change the width, we set $np = \text{const}$. Under this constraint, and assuming $n \gg 1$, the distance (3.35) is minimized when $p_* \approx \sqrt{np/4d}$.

In Figure 3.8b we compare this approximation with the GP kernel computed empirically at initialization. We find good agreement with the theoretical prediction (3.35) when $dp \gg 1$. Furthermore, we see that the minimal kernel distance at initialization and the optimal performance of the trained network are obtained at a similar width, providing evidence in support of our hypothesis.

3.6 Discussion

In this work we studied the question: Do wider networks perform better because they have more parameters, or because of the larger width itself? We considered several ways of increasing the width while keeping the number of parameters fixed, either by introducing bottlenecks into the network, or by sparsifying the weight tensors using a static, random mask. Among the methods we tested, the one that provided the cleanest approach was

removing weights at random in proportion to the layer size, using a static mask generated at initialization. In our image classification experiments, increasing the width using this sparsity method (while keeping the total number of parameters constant) led to significant improvements in model performance. The effect was strongest when starting with a narrow baseline model. Additionally, when comparing the wide, sparse models against dense models of the same width, we found that the width itself accounts for most of the performance gains; this holds true up to the width above which the training accuracy of the sparse models begins to deteriorate, presumably due to low connectivity between the layers.

Focusing on the sparsity method, we initiated a theoretical study of the effect, hypothesizing that the improvement in performance is correlated with having a Gaussian Process kernel that is closer to the infinite-width kernel. We computed the GP kernel of a sparse, 2-layer ReLU network, and derived a simple approximate formula for the distance between this kernel and the infinite-width dense kernel. In our experiment, we found surprisingly strong correlation between the model performance and the distance to the infinite-width kernel.

While our work is fundamental in nature, and sparsity is not the subject of this paper, the method we propose may lead to practical benefits in the future. Using current hardware and available deep learning libraries, we cannot reap the benefits of a sparse weight tensor in terms of reduced computational budget. However, in our experiments we find that the optimal sparsity can be around 1-10% for convolutional models (corresponding to a widening factor of between 3-10). Therefore, using an implementation that natively supports sparse operations, our method may be used to build faster, more memory-efficient networks.

Chapter 4

Restricted Boltzmann Machines

This chapter has been published online as part of the QuCumber software package documentation.

The previous chapters focused on the theory of and the fundamental open questions in DL. Specifically, the work presented in chapter 3 investigated a problem related to over-parametrization, and demonstrated a combination of experimental and theoretical analyses as a successful strategy for the study of deep NNs. All experiments in that study were carried out on the discriminative task of image classification, using standard modern NN model architectures and computer vision datasets.

The current and the following three chapters represent a different line of research, as they are dedicated primarily to ML for physical systems. The experiments and approaches thus differ from the previous ones in several respects. Most importantly, the following studies are situated in an entirely different domain of learning – namely, here we are working with *generative modeling*, as opposed to classification tasks which fall under *discriminative* learning. In particular, we investigate aspects of generative modeling using Restricted Boltzmann Machines (RBMs). Generative modeling is a branch of *unsupervised learning*, while *discriminative* learning considered in the previous chapter is a case of *supervised* learning. In the generative setting, the goal of learning is to enable the model to produce outputs from the same distribution as the training inputs. In essense, a generative model is thought of as a NN-parametrized ansatz for the unknown target distribution. The fundamental assumption underlying generative modeling is that the inputs contained in the training set are all drawn from this target distribution.

RBMs are a prime example of NN models that are not deep; as such, they are not part of applied ML – at most, RBMs can sometimes serve as building blocks in deep

architectures. However, RBMs still play an important role in academical ML. The RBM was first introduced by Smolensky (1986) under the name *"Harmonium"* and was meant to model certain functions of the human brain. However, it became a practical ML model only 15 years later when Hinton (2002) found an algorithm called *Contrastive Divergence* that facilitated efficient training of RBMs. Later, Le Roux and Bengio (2008) have proven that an RBM is an *universal approximator* of discrete distributions. In particular, their paper showed that adding a single hidden unit strictly improves the modeling power as it increases the log-likelihood of data, and thus with enough hidden units one can model any discrete distribution. For more historical context, I recommend reading the respective sections in Giacomo Torlai's thesis (Torlai, 2018). The appeal of RBMs for those studying ML is that these NN models have a solid theoretical foundation and display a powerful ability in generative modeling despite the fact that, on the scale of modern DL, they are small and comparably simple. In this sense, RBMs are situated right at the transition point between old-school ML and modern DL.

In this chapter, I provide an introduction to RBMs in general (mainly based on Fischer and Igel (2011); Hinton (2012)), in preparation for the following three chapters that all employ the RBM for learning the distribution underlying the state of a quantum spin system. The setup is discussed in detail in the respective chapters.

4.1 Definitions

RBMs are among the top methods in unsupervised machine learning, where the training data are inputs without labels and the task is, broadly speaking, to extract some meaningful information from this data. An RBM is a parametrized generative model representing a probability distribution. The training data is assumed to be a sample drawn independently from an unknown **target distribution** q. The goal of training is to fit the parameters λ of the RBM's distribution p_λ such that it resembles the target distribution q as accurately as possible.

Formally, RBMs belong to the class of *undirected graphical models*, also known as *Markov Random Fields* (MRFs). Probabilistic graphical models describe probability distributions by mapping conditional (in)dependence properties between random variables on a graph structure. Visualization by graphs is useful for understanding and motivating probabilistic models. Moreover, it can be helpful for deriving complex computations by using algorithms that exploit the graph structure.

Practically, an RBM is a two-layer network with bidirectionally connected stochastic processing units, as shown in figure 4.1. The V units in the first layer, denoted by the vector

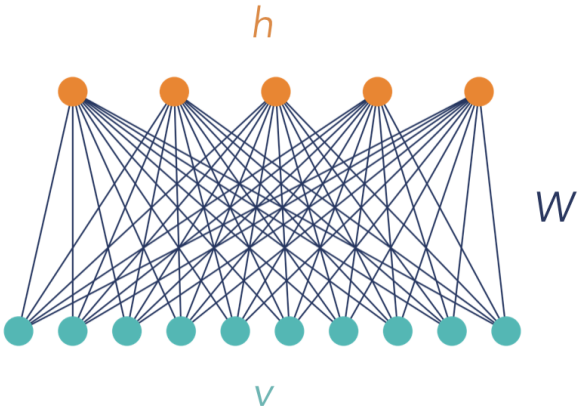


Figure 4.1: RBM as a bipartite graph, with hidden units h, visible units v, and connecting weights W.

$v = (v_1, ..., v_V)$, correspond to the components of an observation and are therefore called *"visible"*, while the H units in the second layer $h = (h_1, ..., h_H)$ represent latent variables, and are referred to as *"hidden"*. Hidden units model dependencies between the observation components and can be viewed as feature detectors. The term *"restricted"* refers to the connections between the units: Each visible unit is connected with each hidden unit, but there are no connections between units of the same kind. In the simplest case (which is the case considered here), all units are binary, such that $v \in \mathcal{V} = \{0,1\}^V$ and $h \in \mathcal{H} = \{0,1\}^H$, where the curved letters $\mathcal{V}$ and $\mathcal{H}$ are used to denote the space of the visible and hidden vectors, respectively. Further, we use $\mathcal{D}$ to denote the training set containing $|\mathcal{D}|$ instances of the visible vector (i.e., observations).

In analogy to spin models in statistical physics, the joint configuration (v, h) of visible and hidden units is characterized by an **energy**

$$\begin{aligned} E_\lambda(v,h) &= -b^\top v - c^\top h - h^\top W v \\ &= -\sum_{i=1}^{V} b_i v_i - \sum_{j=1}^{H} c_j h_j - \sum_{ij} v_i W_{ij} h_j \,, \end{aligned} \tag{4.1}$$

where v_i, h_j are binary states of visible unit i and hidden unit j, and b_i, c_j are their respective **biases**. W_{ij} is the symmetric connection **weight** between the units. The complete **set of parameters** is denoted by $\lambda = \{b, c, W\}$. All parameter values are real numbers.

Based on this energy function, the RBM assigns a probability to each joint configuration

(v, h), which by convention is high when the energy of the configuration is low:

$$p_\lambda(v, h) = \frac{1}{Z} e^{-E_\lambda(v,h)} \,, \tag{4.2}$$

known as the *Gibbs distribution*.

The RBM models the **probability distribution of an input vector** v, $p_\lambda(v)$, which is formally obtained by marginalizing the joint probability distribution $p_\lambda(v, h)$ over all possible hidden vectors h:

$$p_\lambda(v) \;=\; \sum_h p_\lambda(v, h) \;=\; \frac{1}{Z} \sum_h e^{-E_\lambda(v,h)} \;\equiv\; \frac{1}{Z} e^{-\mathcal{E}_\lambda(v)} \,, \tag{4.3}$$

where I have introduced the **effective energy** $\mathcal{E}_\lambda(v)$.

The normalization factor Z, or **partition function**, is obtained by summing up the Boltzmann factors for all possible configurations of the visible and hidden vectors:

$$Z = \sum_{v \in \mathcal{V}} \sum_{h \in \mathcal{H}} e^{-E_\lambda(v,h)} = \sum_v e^{-\mathcal{E}_\lambda(v)} \,. \tag{4.4}$$

The **effective energy** is defined as

$$\mathcal{E}_\lambda(v) = -b^\top v - \sum_{j=1}^{H} \ln \left[1 + \exp\left(c_j + \sum_i W_{ij} v_i \right) \right] \,. \tag{4.5}$$

Note that in some ML references this quantity is called the "free energy", but it is not the same as the free energy in physics, which is defined the partition function:

$$F = -k_B T \ln Z \qquad \Leftrightarrow \qquad Z = e^{-F/(k_B T)} = e^{-\beta F} \tag{4.6}$$

The joint probability distribution is related to the conditional distributions via the chain rule as $p_\lambda(v, h) = p_\lambda(v|h) p_\lambda(h) = p_\lambda(h|v) p_\lambda(v)$. Because there are no direct connections between units of the same layer in an RBM, the **conditional distributions** $p_\lambda(h|v)$ and $p_\lambda(v|h)$ factorize over each unit and are easy to compute. With some straightforward algebra, one can show that

$$p_\lambda(h|v) = \prod_{j=1}^{H} p_\lambda(h_j|v) \,, \tag{4.7}$$

$$p_\lambda(v|h) = \prod_{i=1}^{V} p_\lambda(v_i|h) \,, \tag{4.8}$$

and

$$p_\lambda(h_j = 1|v) = \mathcal{S}\left(c_j + \sum_i v_i W_{ij}\right) , \tag{4.9}$$

$$p_\lambda(v_i = 1|h) = \mathcal{S}\left(b_i + \sum_j h_j W_{ij}\right) , \tag{4.10}$$

with $\mathcal{S}$ denoting the *standard logistic* sigmoid function:

$$\mathcal{S}(x) = \frac{1}{1 + e^{-x}} . \tag{4.11}$$

4.2 Training

Training the RBM means adjusting parameters λ based on the given data set $\mathcal{D}$, such that $p_\lambda(v)$ is a good approximation of the true distribution $q(v)$. In the supervised setting, we would define a cost function that measures the discrepancy between the network prediction and the desired output (which is part of the training set), and perform a minimization of the cost function. Similarly, in the unsupervised setting we introduce a **cost function** C_λ that measures how the probability distribution p_λ predicted by the RBM is different from q, known as the *Kullback-Leibler (KL) divergence* or the *relative entropy*:

$$C_\lambda = D_{KL}(q\|p_\lambda) = \sum_v q(v) \ln \frac{q(v)}{p_\lambda(v)} = -H(q) - \langle \ln p_\lambda(v) \rangle_q . \tag{4.12}$$

In the expression on the right-hand side, the first term is simply the *Shannon entropy* of q,

$$H(q) = -\sum_v q(v) \ln q(v), \tag{4.13}$$

and the second term is an expectation value of the quantity $\ln p_\lambda(v)$ called the *log-likelihood*:

$$\langle \ln p_\lambda(v) \rangle_q = \sum_v q(v) \ln p_\lambda(v). \tag{4.14}$$

The goal of the training procedure is then to find a set of parameters λ that minimizes the cost function C_λ. The minimization is performed by *gradient descent*, and the **update rule** for the parameters has the familiar form

$$\lambda \leftarrow \lambda - \eta \nabla_\lambda C_\lambda, \tag{4.15}$$

where η is the *learning rate*.

Note that $H(q)$ does not depend on λ, such that only the log-likelihood term is relevant for the optimization:

$$\nabla_\lambda C_\lambda = -\nabla_\lambda \langle \ln p_\lambda(v) \rangle_q. \tag{4.16}$$

However, the log-likelihood term involves the unknown target distribution q. In order to proceed with the minimization, we approximate q by the **empirical distribution** of the training data $q_\mathcal{D}$:

$$q(v) \simeq q_\mathcal{D}(v) = \frac{1}{|\mathcal{D}|} \sum_{\tilde{v} \in \mathcal{D}} \delta(v - \tilde{v}). \tag{4.17}$$

The expectation value in eq. (4.16) can then be evaluated as follows:

$$\begin{aligned} \langle \ln p_\lambda(v) \rangle_q \simeq \langle \ln p_\lambda(v) \rangle_\mathcal{D} &= \frac{1}{|\mathcal{D}|} \sum_v \sum_{\tilde{v} \in \mathcal{D}} \delta(v - \tilde{v}) \ln p_\lambda(v) \\ &= \frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} \ln p_\lambda(v) \\ &= -\ln Z_\lambda - \frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} \mathcal{E}_\lambda(v), \end{aligned} \tag{4.18}$$

where I have substituted the expression for $p_\lambda(v)$ from (4.3). Thus, the gradient of the cost function is

$$\begin{aligned} \nabla_\lambda C_\lambda \simeq -\nabla_\lambda \langle \ln p_\lambda(v) \rangle_\mathcal{D} &= \frac{1}{|\mathcal{D}|} \sum_{v \in \mathcal{D}} \nabla_\lambda \mathcal{E}_\lambda(v) + \nabla_\lambda \ln Z_\lambda \\ &= \sum_v q_\mathcal{D}(v) \nabla_\lambda \mathcal{E}_\lambda(v) - \sum_v p_\lambda(v) \nabla_\lambda \mathcal{E}_\lambda(v) \\ &= \langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_\mathcal{D} - \langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}. \end{aligned} \tag{4.19}$$

We evaluate $\nabla_\lambda \mathcal{E}_\lambda$ for each parameter in λ by computing the gradient element-wise:

$$\frac{\partial \mathcal{E}_\lambda(v)}{\partial W_{ij}} = -v_i \, \mathcal{S}\left(c_j + \sum_k v_k W_{kj} \right) = -v_i \, p_\lambda(h_j = 1 | v) \tag{4.20}$$

$$\frac{\partial \mathcal{E}_\lambda(v)}{\partial c_j} = -\mathcal{S}\left(c_j + \sum_i v_i W_{ij} \right) = -p_\lambda(h_j = 1 | v) \tag{4.21}$$

$$\frac{\partial \mathcal{E}_\lambda(v)}{\partial b_i} = -v_i \tag{4.22}$$

The expectation value over the empirical probability distribution, $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{\mathcal{D}}$, can easily be computed based on the training set. The term $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}$, however, poses some serious problems. This expectation value is over the marginalized probability distribution $p_\lambda(v) = e^{-\mathcal{E}_\lambda(v)}/Z$ and involves the partition function Z which requires evaluation of the sum over all $h \in \mathcal{H}$ **and** $v \in \mathcal{V}$, and is thus in general intractable. There are several possible approximate methods to handle this term, but not all of them are practicable. The straightforward approach is to approximate the expectation $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}$ by an *estimator* sampled from the model distribution p_λ using *Gibbs sampling*. In Gibbs sampling, each variable is sampled from its conditional distribution given the current states of the other variables. Starting from a randomly initialized visible state $v = v^{(0)}$, we alternate between updating h and v according to $p_\lambda(h|v)$ and $p_\lambda(v|h)$. In RBMs, this procedure is particularly efficient because the visible units are conditionally independent given the hidden units (and vice versa), such that the calculation can be carried out for all units in parallel, as illustrated in figure 4.2. This is referred to as *Block Gibbs Sampling*. We repeat this procedure on M different initial states and use the average to approximate the expectation value:

$$\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda} = \sum_v p_\lambda(v) \nabla_\lambda \mathcal{E}_\lambda(v) \simeq \frac{1}{M} \sum_{v^{(t)} \in \mathcal{M}} \nabla_\lambda \mathcal{E}_\lambda(v^{(t)}), \tag{4.23}$$

where the index t refers to the number of Gibbs steps performed and $\mathcal{M}$ denotes the set of M visible state vectors $v^{(t)}$.

However, to ensure that the Markov chain converges to stationarity, the sampling process has to be run for a long time, i.e., the number of Gibbs steps t has to be large. Since this process has to be repeated for each parameter update in the learning procedure, this technique is computationally not feasible. Therefore, all methods that are employed in practice introduce additional approximations.

The standard **learning algorithm** employed for RBM training is called *Contrastive Divergence* (CD-k) (Hinton, 2002). Essentially, the idea of this algorithm is to perform only k steps of Gibbs sampling starting from a current training vector $v = v^{(0)}$[1]. Even though it is only crudely approximating the true expectation, the learning works surprisingly well. In general, larger k yields a less biased estimate; however, $k = 1$ is often sufficient to extract meaningful features in practice.

To sum up, the main steps of the CD-k learning procedure are:

0. Select M vectors from the training data (referred to as *mini batch*) and perform the next two steps for each of the M vectors in parallel.

[1] In contrast, in standard Gibbs sampling, one starts from a randomly initialized vector.

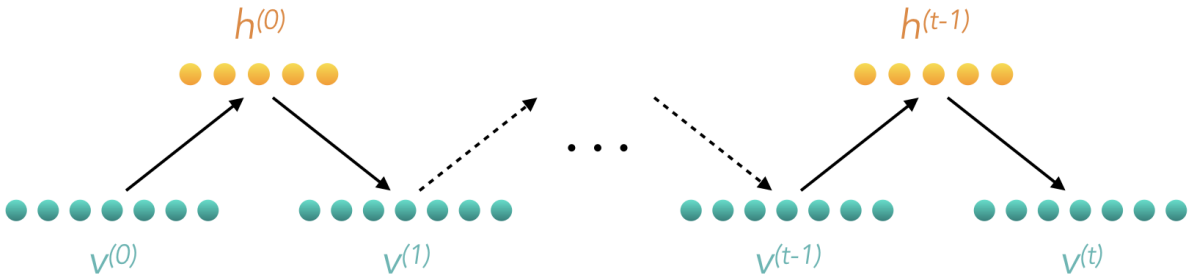


Figure 4.2: In RBMs, the visible and hidden units are conditionally independent of each other, such that Gibbs sampling can alternate between parallel updates of the hidden and visible units. At $t \to \infty$, the samples $(v^{(t)}, h^{(t)})$ are guaranteed to be accurate samples of the model distribution $p_\lambda(v, h)$.

1. Initialize by setting the visibles to the training vector v.
2. Perform the following Gibbs sampling procedure k times:
   - Compute the states of the hiddens (in parallel) by setting each unit to 1 with a probability given by eq. (4.9).
   - Produce a *"reconstruction"* of v (denoted as $v^{(i)}$, where i is the iteration step number), by setting each visible to 1 with a probability given by eq. (4.8).
3. Use the M reconstructions $v^{(k)}$ to compute the estimator for $\langle \nabla_\lambda \mathcal{E}_\lambda(v) \rangle_{p_\lambda}$ and update the model parameters λ.
4. Go back to step 0 and repeat the procedure until stopping criteria fulfilled.

The CD-k update rule for the *weights* can be written as

$$W \leftarrow W + \eta \left[\mathcal{S}\left(b + Wv\right) v^\top - \mathcal{S}\left(b + W\tilde{v}\right) \tilde{v}^\top \right],$$

with a training example $v \in \mathcal{D}$ and the so-called *"negative sample"* $\tilde{v}$, obtained from v by Gibbs sampling. A simplified version of the same learning rule is applied to the *biases*; it involves the states of individual units instead of pairwise products:

$$a \leftarrow a + \eta \left(v - \tilde{v}\right),$$

and

$$b \leftarrow b + \eta \left[\mathcal{S}\left(b + Wv\right) - \mathcal{S}\left(b + W\tilde{v}\right) \right].$$

Note that some references define a short-hand notation

$$\mathcal{S}\left(b + Wv\right) \equiv h(v),$$

because the logistic function essentially defines the state of h. However, it is important to keep in mind that h is a binary vector, and $\mathcal{S}\left(b + Wv\right)$ does not provide the values, but the *probability* for each element of h having the value 1. This actually leads to the following practical remark:

A remark on computing the state of a binary unit

Given a visible vector v and the set of parameters λ, how do we compute the state of a hidden unit h_j? Since the unit is binary, its state is sampled according to the *Bernoulli distribution* with the probability for h_j having the value 1 given by eq. (4.9). In practice, we sample a number u from a *uniform distribution* over the interval $[0, 1]$, $u \sim U[0, 1]$. If $u < p_\lambda(h_j = 1|v)$, set $h_j = 1$, otherwise $h_j = 0$.

A slight modification of the CD algorithm leads to the *Persistent Contrastive Divergence* (PCD) algorithm (Tieleman, 2008). In PCD, instead of initializing the chain to $v^{(0)} = v \in \mathcal{D}$ each time, we use the negative sample from the previous iteration. That is, PCD just keeps the Markov chain evolving, with parameter updates done after each k steps. The number of persistent chains used for sampling is a hyperparameter. In the standard case, there is one Markov chain per training example in a batch.

4.3 Parametrization

In the standard definition, the values of the hidden and visible units in the RBM are in $\{0, 1\}$. In some cases, it can be convenient to work with values in $\{\pm 1\}$. The reparametrization from $v_i, h_j \in \{0, 1\}$ to $\tilde{v}_i, \tilde{h}_j \in \{\pm 1\}$

$$\begin{aligned} v_i = \frac{1}{2}(\tilde{v}_i + 1) \quad &\Leftrightarrow \quad \tilde{v}_i = 2v_i - 1, \\ h_j = \frac{1}{2}(\tilde{h}_j + 1) \quad &\Leftrightarrow \quad \tilde{h}_j = 2h_j - 1 \end{aligned}$$

amounts to a simple parameter mapping in the RBM energy function:

$$\tilde{b}_i = \frac{1}{4}\left(2b_i + \sum_j W_{ij}\right), \qquad \tilde{c}_j = \frac{1}{4}\left(2c_j + \sum_i W_{ij}\right), \qquad \tilde{W}_{ij} = \frac{1}{4}W_{ij}\,, \tag{4.24}$$

and a constant term $\tilde{Q}$:

$$\tilde{Q} = \sum_i \tilde{b}_i + \sum_j \tilde{c}_j - \sum_{ij} \tilde{W}_{ij} \tag{4.25}$$

$$= \frac{1}{2}\sum_i b_i + \frac{1}{2}\sum_j c_j + \frac{1}{4}\sum_{ij} W_{ij}\,. \tag{4.26}$$

4.4 Final remark

The following chapter demonstrates the implementation of an RBM for the task of quantum state reconstruction.

Chapter 5

Quantum State Reconstruction with RBMs

The work presented in this chapter has been published as QuCumber.

In the previous chapter I provided an introduction to Restricted Boltzmann Machines (RBMs) and their function as generative models in ML. The current chapter is the first in a series of research projects that explore generative modeling with RBMs for physical quantum systems. More specifically, the goal is to learn to model the distribution underlying the states of a quantum spin chain. The training set is composed of projective measurements of a physical quantum system that can in principle be a result from an experiment performed in a lab, or simulated numerically. In theoretical investigations, we consider spin configurations obtained from numerical simulations of a quantum spin model in different coupling regimes, such as the transverse-field Ising model. The RBM-based method is applicable in case when the quantum wavefunction is real and positive, but can also be extended to the more general case of complex-valued wavefunctions. Both cases are dicussed in this chapter.

The character of this chapter is rather technical, as it is concerned with the *implementation* of RBMs for the purpose of quantum state reconstruction in Python and was written as a supplement to the open-source software package QuCumber. Its main purpose is to provide a comprehensive user guide to the software, and therefore it walks the reader through all applications with code examples. It also provides a brief introduction to the problem of quantum state reconstruction and motivates the application of generative modeling with RBMs as a possible efficient solution.

The QuCumber software package forms the technical base for two research projects,

presented in the following two chapters, that focus on the study of various aspects of RBM-based learning for quantum systems.

5.1 Abstract

As we enter a new era of quantum technology, it is increasingly important to develop methods to aid in the accurate preparation of quantum states for a variety of materials, matter, and devices. Computational techniques can be used to reconstruct a state from data, however, the growing number of qubits demands ongoing algorithmic advances in order to keep pace with experiments. In this work, we present an open-source software package called *QuCumber* that uses machine learning with an RBM to reconstruct a quantum state consistent with a set of projective measurements. The RBM can efficiently represent the quantum wavefunction for a large number of qubits. New measurements can be generated from the trained machine to obtain physical observables not easily accessible from the original data.

5.2 Motivation

Current advances in quantum technologies, as well as in reliable control of synthetic quantum matter, are leading to a new era of quantum hardware where highly pure quantum states are routinely prepared in laboratories. With the growing number of controlled quantum degrees of freedom, such as superconducting qubits, trapped ions, and ultracold atoms (Kandala et al., 2017; Moll et al., 2018; Bernien et al., 2017; Zhang et al., 2017b), reliable and scalable classical algorithms are required for the analysis and verification of experimentally prepared quantum states. Efficient algorithms can aid in extracting physical observables otherwise inaccessible from experimental measurements, as well as in identifying sources of noise to provide direct feedback for improving experimental hardware. However, traditional approaches for reconstructing unknown quantum states from a set of measurements, such as quantum state tomography, often suffer the exponential overhead that is typical of quantum many-body systems.

Recently, an alternative path to quantum state reconstruction was put forward, based on modern machine learning (ML) techniques (Torlai and Melko, 2016; Torlai et al., 2018; Torlai and Melko, 2018; Carrasquilla et al., 2019; Lennon et al., 2018; Kim et al., 2018). The most common approach relies on a powerful generative model called a *Restricted Boltzmann Machine* (RBM) (Smolensky, 1986) – a stochastic neural network with two layers of binary

units. A visible layer $\boldsymbol{v}$ describes the physical degrees of freedom, while a hidden layer $\boldsymbol{h}$ is used to capture high-order correlations between the visible units. Given a set of neural network parameters $\boldsymbol{\lambda}$, the RBM defines a probabilistic model described by the parametric distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{v})$. RBMs have been widely used in the ML community for the pre-training of deep neural networks (Hinton, 2002), for compressing high-dimensional data into lower-dimensional representations (Hinton and Salakhutdinov, 2006), and more (LeCun et al., 2015). More recently, RBMs have been adopted by the physics community in the context of representing both classical and quantum many-body states (Carleo and Troyer, 2017a; Carleo et al., 2018a). They are currently being investigated for their representational power (Gao and Duan, 2017a; Choo et al., 2018; Glasser et al., 2018a), their relationship with tensor networks and the renormalization group (Mehta and Schwab, 2014; Koch-Janusz and Ringel, 2018; Iso et al., 2018; Lenggenhager et al., 2018; Chen et al., 2018), and in other contexts in quantum many-body physics (Nomura et al., 2017; Weinstein, 2018; Zheng et al., 2018).

In this post, we present *QuCumber: a quantum calculator used for many-body eigenstate reconstruction*. QuCumber is an open-source Python package that implements neural-network quantum state reconstruction of many-body wavefunctions from projective measurement data. Examples of data to which QuCumber could be applied might be magnetic spin projections, orbital occupation number, polarization of photons, or the logical state of qubits. Given a training set of such measurements, QuCumber discovers the most likely compatible quantum state by finding the optimal set of parameters $\boldsymbol{\lambda}$ of an RBM. A properly trained RBM is an approximation of the unknown quantum state underlying the data. It can be used to calculate various physical observables of interest, including measurements that may not be possible in the original experiment.

This work is organized as follows: In Section 5.3, we introduce the reconstruction technique for the case where all coefficients of the wavefunction are *real* and *positive*. We discuss the required format for input data, as well as training of the RBM and the reconstruction of typical observables. In Section 5.4, we consider the more general case of a *complex-valued* wavefunction. We illustrate a general strategy to extract the phase structure from data by performing appropriate unitary rotations on the state before measurements. We then demonstrate a practical reconstruction of an entangled state of two qubits. Note, the detailed theory underlying the reconstruction methods used by QuCumber can be found in the original references (Torlai and Melko, 2016; Torlai et al., 2018) and a recent review by Torlai and Melko (2020).

5.3 Positive wavefunctions

We begin by presenting the application of QuCumber to reconstruct many-body quantum states described by wavefunctions $|\Psi\rangle$ with positive coefficients $\Psi(\boldsymbol{x}) = \langle \boldsymbol{x}|\Psi\rangle \geq 0$, where $|\boldsymbol{x}\rangle = |x_1, \ldots, x_N\rangle$ is a reference basis for the Hilbert space of N quantum degrees of freedom. The neural-network quantum state reconstruction requires raw data $\mathcal{D} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots)$ generated through projective measurements of the state $|\Psi\rangle$ in the reference basis. These measurements adhere to the probability distribution given by the Born rule, $P(\boldsymbol{x}) = |\Psi(\boldsymbol{x})|^2$. Since the wavefunction is strictly positive, the quantum state is completely characterized by the measurement distribution, i.e. $\Psi(\boldsymbol{x}) = \sqrt{P(\boldsymbol{x})}$.

The positivity of the wavefunction allows a simple and natural connection between quantum states and classical probabilistic models. QuCumber employs the probability distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{x})$ of an RBM (see Eq. 4.3) to approximate the distribution $P(\boldsymbol{x})$ underlying the measurement data. Using contrastive divergence (CD) (Hinton, 2002), QuCumber trains the RBM to discover an optimal set of parameters $\boldsymbol{\lambda}$ that minimize the Kullback-Leibler (KL) divergence between the two distributions (see Eq. 4.12). Upon successful training ($p_{\boldsymbol{\lambda}}(\boldsymbol{x}) \sim P(\boldsymbol{x})$), we obtain an approximate representation of the target quantum state,

$$\psi_{\boldsymbol{\lambda}}(\boldsymbol{x}) \equiv \sqrt{p_{\boldsymbol{\lambda}}(\boldsymbol{x})} \simeq \Psi(\boldsymbol{x}) \,. \tag{5.1}$$

Note, the precise mathematical form of the marginal distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{x})$ defined in terms of an effective energy over the parameters of the RBM is defined in the Glossary.

In the following, we demonstrate the application of QuCumber for the reconstruction of the ground-state wavefunction of the one-dimensional transverse-field Ising model (TFIM). The Hamiltonian is

$$\hat{H} = -J \sum_i \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h \sum_i \hat{\sigma}_i^x \,, \tag{5.2}$$

where $\hat{\sigma}_i^{x/z}$ are spin-1/2 Pauli operators acting on site i, and we assume open boundary conditions. For this example, we consider a chain with $N = 10$ spins at the quantum critical point $J = h = 1$.

5.3.1 Setup

Given the small size of the system, the ground state $|\Psi\rangle$ can be found with exact diagonalization. The training dataset $\mathcal{D}$ is generated by sampling the distribution $P(\boldsymbol{\sigma}^z) = |\Psi(\boldsymbol{\sigma}^z)|^2$, obtaining a sequence of $N_S = 10^5$ independent spin projections in the reference

basis $\boldsymbol{x} = \boldsymbol{\sigma}^z$.[1] Each data point in $\mathcal{D}$ consists of an array $\boldsymbol{\sigma}_j^z = (\sigma_1^z, \ldots, \sigma_N^z)$ with shape `(N,)` and should be passed to QuCumber as a numpy array or torch tensor. For example, $\boldsymbol{\sigma}_j^z =$ `np.array([1,0,1,1,0,1,0,0,0,1])`, where we use $\sigma_j^z = 0, 1$ to represent a spin-down and spin-up state respectively. Therefore, the entire input data set is contained in an array with shape `(N_S, N)`.

Aside from the training data, QuCumber also allows us to import an exact wavefunction. This can be useful for monitoring the quality of the reconstruction during training. In our example, we evaluate the fidelity between the reconstructed state $\psi_{\boldsymbol{\lambda}}(\boldsymbol{x})$ and the exact wavefunction $\Psi(\boldsymbol{x})$. The training dataset, `train_data`, and the exact ground state, `true_psi`, are loaded with the data loading utility as follows:

```
import qucumber.utils.data as data
train_path = "tfim1d_data.txt"
psi_path = "tfim1d_psi.txt"
train_data, true_psi = data.load_data(train_path, psi_path)
```

If `psi_path` is not provided, QuCumber will load only the training data.

Next, we initialize an RBM quantum state $\psi_{\boldsymbol{\lambda}}(\mathbf{x})$ with random weights and zero biases using the constructor `PositiveWaveFunction`:

```
from qucumber.nn_states import PositiveWaveFunction
state = PositiveWaveFunction(num_visible=10, num_hidden=10)
```

The number of visible units (`num_visible`) must be equal to the number of physical spins N, while the number of hidden units (`num_hidden`) can be adjusted to systematically increase the representational power of the RBM.

The quality of the reconstruction will depend on the structure underlying the specific quantum state and the ratio of visible to hidden units, $\alpha =$ `num_hidden`/`num_visible`. In practice, we find that $\alpha = 1$ often leads to good approximations of positive wavefunctions (Torlai et al., 2018). However, in the general case, the value of α required for a given wavefunction should be explored and adjusted by the user.

5.3.2 Training

Once an appropriate representation of the quantum state has been defined, QuCumber trains the RBM through the function `PositiveWaveFunction.fit`. Several input param-

[1] The training dataset can be downloaded from https://github.com/PIQuIL/QuCumber/blob/master/examples/Tutorial1_TrainPosRealWaveFunction/tfim1d_data.txt

eters need to be provided aside from the training dataset (`train_data`). These include the number of training iterations (`epochs`), the number of samples used for the positive/negative phase of CD (`pos_batch_size`/`neg_batch_size`), the learning rate (`lr`) and the number of sampling steps in the negative phase of CD (`k`). The last argument (`callbacks`) allows the user to pass a set of additional functions to be evaluated during training.

As an example of a callback, we show the `MetricEvaluator`, which evaluates a function `log_every` epochs during training. Given the small system size and the knowledge of the true ground state, we can evaluate the fidelity between the RBM state and the true ground-state wavefunction (`true_psi`). Similarly, we can calculate the KL divergence between the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{x})$, and the data distribution $P(\boldsymbol{x})$, which should approach zero for a properly trained RBM. For the current example, we monitor the fidelity and KL divergence (defined in `qucumber.utils.training_statistics`):

```
from qucumber.callbacks import MetricEvaluator
import qucumber.utils.training_statistics as ts
log_every = 10
space = state.generate_hilbert_space(10)
callbacks = [
    MetricEvaluator(
        log_every,
        {"Fidelity": ts.fidelity, "KL": ts.KL},
        target_psi=true_psi,
        space=space,
        verbose=True
    )]
```

With `verbose=True`, the program will print the epoch number and all callbacks every `log_every` epochs. For the current example, we monitor the fidelity and KL divergence. Note that the KL divergence is only tractable for small systems. The `MetricEvaluator` will compute the KL exactly when provided with a list of all states in the Hilbert space. For convenience these can be generated with
`space = state.generate_hilbert_space(10)`.

Now that the metrics to monitor during training have been chosen, we can invoke the optimization with the `fit` function of `PositiveWaveFunction`.

```
state.fit(
    train_data,
    epochs=500,
    pos_batch_size=100,
```

```
    neg_batch_size=100,
    lr=0.01,
    k=5,
    callbacks=callbacks,
)
```

Figure 5.1 shows the convergence of the fidelity and KL divergence during training. The convergence time will, in general, depend on the choice of hyperparameters. Finally, the network parameters $\boldsymbol{\lambda}$, together with the `MetricEvaluator`'s data, can be saved (or loaded) to a file:

```
state.save(
    "filename.pt",
    metadata={
        "fidelity": callbacks[0].Fidelity,
        "KL": callbacks[0].KL
    },
)
state.load("filename.pt")
```

With this we have demonstrated the most basic aspects of QuCumber regarding training a model and verifying its accuracy. We note that in this example the evaluation utilized the knowledge of the exact ground state and the calculation of the KL divergence, which we reemphasize is tractable only for small system sizes. However, we point out that QuCumber is capable of carrying out the reconstruction of much larger systems. In such cases, users must rely on other estimators to evaluate the training, such as expectation values of physical observables (magnetization, energy, etc). In the following, we show how to compute diagonal and off-diagonal observables in QuCumber.

5.3.3 Reconstruction of physical observables

In this section, we discuss how to calculate the average value of a generic physical observable $\hat{\mathcal{O}}$ from a trained RBM. We start with the case of observables that are diagonal in the reference basis where the RBM was trained. We then discuss the more general cases of off-diagonal observables and entanglement entropies.

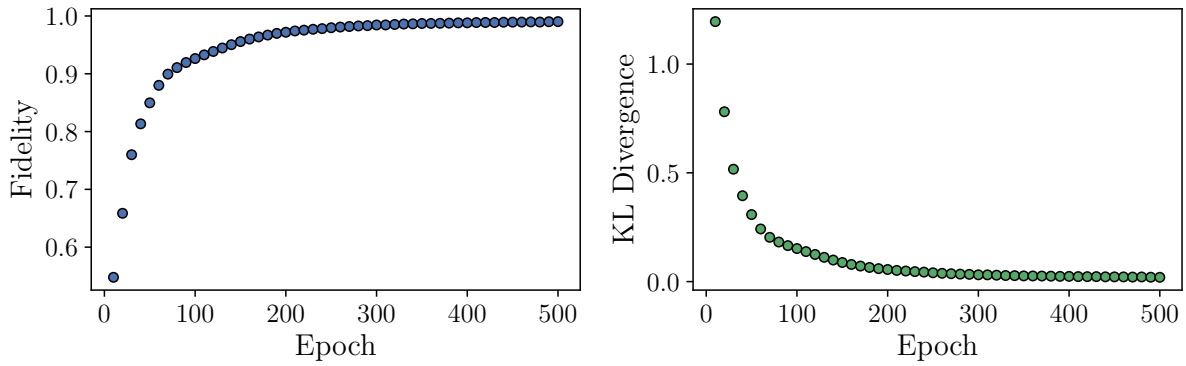


Figure 5.1: The fidelity (left) and the KL divergence (right) during training for the reconstruction of the ground state of the one-dimensional TFIM.

Diagonal observables

We begin by considering an observable with only diagonal matrix elements, $\langle \boldsymbol{\sigma} | \hat{\mathcal{O}} | \boldsymbol{\sigma}' \rangle = \mathcal{O}_{\boldsymbol{\sigma}} \delta_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}$ where for convenience we denote the reference basis $\mathbf{x} = \boldsymbol{\sigma}^z$ as $\boldsymbol{\sigma}$ unless otherwise stated. The expectation value of $\hat{\mathcal{O}}$ is given by

$$\langle \hat{\mathcal{O}} \rangle = \frac{1}{\sum_{\boldsymbol{\sigma}} |\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})|^2} \sum_{\boldsymbol{\sigma}} \mathcal{O}_{\boldsymbol{\sigma}} |\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})|^2 \,. \tag{5.3}$$

The expectation value can be approximated by a Monte Carlo estimator,

$$\langle \hat{\mathcal{O}} \rangle \approx \frac{1}{N_{\mathrm{MC}}} \sum_{k=1}^{N_{\mathrm{MC}}} \mathcal{O}_{\boldsymbol{\sigma}_k} \,, \tag{5.4}$$

where the spin configurations $\boldsymbol{\sigma}_k$ are sampled from the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})$. This process is particularly efficient given the bipartite structure of the network which allows the use of block Gibbs sampling.

A simple example for the TFIM is the average longitudinal magnetization per spin, $\langle \hat{\sigma}^z \rangle = \sum_j \langle \hat{\sigma}_j^z \rangle / N$, which can be calculated directly on the spin configuration sampled by the RBM (i.e., the state of the visible layer). The visible samples are obtained with the `sample` function of the RBM state object:

```
samples = state.sample(num_samples=1000, k=10)
```

which takes the total number of samples (`num_samples`) and the number of block Gibbs steps (`k`) as input. Once these samples are obtained, the magnetization can be calculated simply as

```
magnetization = samples.mul(2.0).sub(1.0).mean()
```

where we converted the binary samples of the RBM back into ± 1 spins before taking the mean.

Off-diagonal observables

We turn now to the case of off-diagonal observables, where the expectation value assumes the following form

$$\langle \hat{\mathcal{O}} \rangle = \frac{1}{\sum_{\boldsymbol{\sigma}} |\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})|^2} \sum_{\boldsymbol{\sigma}\boldsymbol{\sigma}'} \psi^*_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}) \psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}') \mathcal{O}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'} \,. \tag{5.5}$$

This expression can once again be approximated with a Monte Carlo estimator

$$\langle \hat{\mathcal{O}} \rangle \approx \frac{1}{N_{\mathrm{MC}}} \sum_{k=1}^{N_{\mathrm{MC}}} \mathcal{O}^{[L]}_{\boldsymbol{\sigma}_k} \tag{5.6}$$

of the so-called *local estimator* of the observable:

$$\mathcal{O}^{[L]}_{\boldsymbol{\sigma}_k} = \sum_{\boldsymbol{\sigma}'} \frac{\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}')}{\psi_{\boldsymbol{\lambda}}(\boldsymbol{\sigma}_k)} \mathcal{O}_{\boldsymbol{\sigma}_k \boldsymbol{\sigma}'} \,. \tag{5.7}$$

As long as the matrix representation $\mathcal{O}_{\boldsymbol{\sigma}\boldsymbol{\sigma}'}$ is sufficiently sparse in the reference basis, the summation can be evaluated efficiently.

As an example, we consider the specific case of the transverse magnetization for the j-th spin, $\langle \hat{\sigma}^x_j \rangle$, with matrix elements

$$\langle \boldsymbol{\sigma} | \hat{\sigma}^x_j | \boldsymbol{\sigma}' \rangle = \delta_{\sigma'_j, 1-\sigma_j} \prod_{i \neq j} \delta_{\sigma'_i, \sigma_i} \,. \tag{5.8}$$

Therefore, the expectation values reduces to the Monte Carlo average of the local observable

$$(\sigma^x_j)^{[L]} = \frac{\psi_{\boldsymbol{\lambda}}(\sigma_1, \ldots, 1-\sigma_j, \ldots, \sigma_N)}{\psi_{\boldsymbol{\lambda}}(\sigma_1, \ldots, \sigma_j, \ldots, \sigma_N)} \,. \tag{5.9}$$

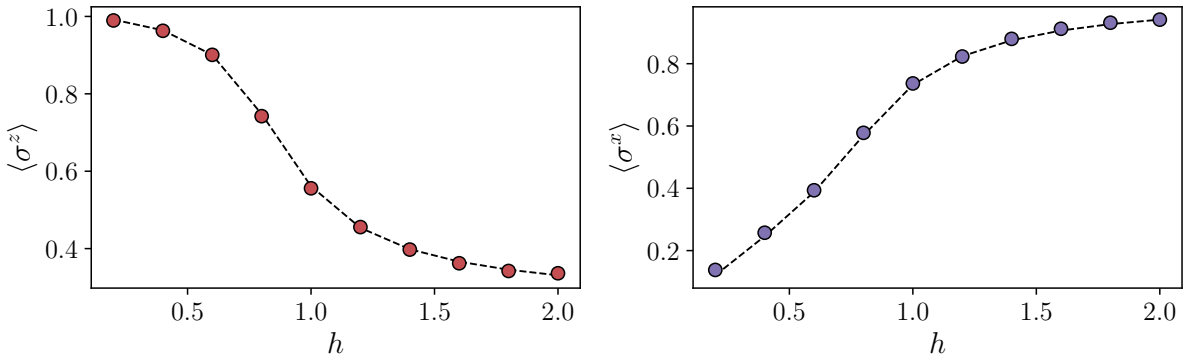


Figure 5.2: Reconstruction of the magnetic observables for the TFIM chain with $N = 10$ spins. We show the average longitudinal (left) and transverse (right) magnetization per site obtained by sampling from a trained RBM. The dashed line denotes the results from exact diagonalization.

evaluated on spin configurations $\boldsymbol{\sigma}_k$ sampled from the RBM distribution $p_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})$.

QuCumber provides an interface for sampling off-diagonal observables in the `ObservableBase` class. Thorough examples are available in the tutorial section in the documentation.[2] As an example, σ^x can be written as an observable class with

```
import torch
from qucumber.utils import cplx
from qucumber.observables import ObservableBase

class SigmaX(ObservableBase):

    def apply(self, nn_state, samples):
        psi = nn_state.psi(samples)
        psi_ratio_sum = torch.zeros_like(psi)

        for i in range(samples.shape[-1]):  # sum over spin sites
            flip_spin(i, samples)  # flip the spin at site i
            # add ratio psi_(-i) / psi to the running sum
            psi_flip = nn_state.psi(samples)
            psi_ratio = cplx.elementwise_division(psi_flip, psi)
```

[2]The observables tutorial is available at https://qucumber.readthedocs.io/en/stable/_examples/Tutorial3_DataGeneration_CalculateObservables/tutorial_sampling_observables.html

```
            psi_ratio_sum.add_(psi_ratio)
            flip_spin(i, samples)  # flip it back

        # take real part and divide by number of spins
        return psi_ratio_sum[0].div_(samples.shape[-1])
```

The value of the observable can be estimated from a set of samples with:

```
SigmaX().statistics_from_samples(state, samples)
```

which produces a dictionary containing the mean, variance, and standard error of the observable. Similarly, the user can define other observables like the energy.

The reconstruction of two magnetic observables for the TFIM is shown in Fig. 5.2, where a different RBM was trained for each value of the transverse field h. In the left plot, we show the average longitudinal magnetization per site, which can be calculated directly from the configurations sampled by the RBM. In the right plot, we show the off-diagonal observable of transverse magnetization. In both cases, QuCumber successfully discovers an optimal set of parameters $\boldsymbol{\lambda}$ that accurately approximate the ground-state wavefunction underlying the data.

Entanglement entropy

A quantity of significant interest in quantum many-body systems is the degree of entanglement between a sub-region A and its complement $\bar{A}$. Numerically, measurement of bipartite entanglement entropy is commonly accessed through the computation of the second Rényi entropy $S_2 = -\ln \mathrm{Tr}(\rho_A^2)$. When one has access to a pure state wavefunction $\psi_{\boldsymbol{\lambda}}(\boldsymbol{x})$, Rényi entropies can be calculated as an expectation value of the "Swap" operator (Hastings et al., 2010),

$$S_2 = -\ln \left\langle \widehat{\mathrm{Swap}}_A \right\rangle . \tag{5.10}$$

It is essentially an off-diagonal observable that acts on an extended product space consisting of two independent copies of the wavefunction, $\psi_{\boldsymbol{\lambda}}(\boldsymbol{x}) \otimes \psi_{\boldsymbol{\lambda}}(\boldsymbol{x})$, referred to as "replicas". As the name suggests, the action of the Swap operator is to swap the spin configurations in region A between the replicas,

$$\widehat{\mathrm{Swap}}_A |\boldsymbol{\sigma}_A, \boldsymbol{\sigma}_{\bar{A}}\rangle_1 \otimes |\boldsymbol{\sigma}'_A, \boldsymbol{\sigma}'_{\bar{A}}\rangle_2 = |\boldsymbol{\sigma}'_A, \boldsymbol{\sigma}_{\bar{A}}\rangle_1 \otimes |\boldsymbol{\sigma}_A, \boldsymbol{\sigma}'_{\bar{A}}\rangle_2 . \tag{5.11}$$

Here the subscript of the ket indicates the replica index, while the two labels inside a ket, such as $\boldsymbol{\sigma}_A, \boldsymbol{\sigma}_{\bar{A}}$, describe the spins configurations within the sub-region and its complement.

In QuCumber, the Swap operator is implemented as a routine within the `entanglement` observable unit,

```
def swap(s1, s2, A):
    _s = s1[:, A].clone()
    s1[:, A] = s2[:, A]
    s2[:, A] = _s
    return s1, s2
```

where `s1` and `s2` are batches of samples produced from each replica, and `A` is a list containing the indices of the sites in sub-region A. While ideally those samples should be entirely independent, in order to save computational costs, QuCumber just splits a given batch into two equal parts and treats them as if they were independent samples. This is implemented within the `SWAP` observable,

```
class SWAP(ObservableBase):
    def __init__(self, A):
        self.A = A

    def apply(self, nn_state, samples):
        _ns = samples.shape[0] // 2
        samples1 = samples[:_ns, :]
        samples2 = samples[_ns : _ns * 2, :]

        psi_ket1 = nn_state.psi(samples1)
        psi_ket2 = nn_state.psi(samples2)

        psi_ket = cplx.elementwise_mult(psi_ket1, psi_ket2)
        psi_ket_star = cplx.conjugate(psi_ket)

        samples1_, samples2_ = swap(samples1, samples2, self.A)
        psi_bra1 = nn_state.psi(samples1_)
        psi_bra2 = nn_state.psi(samples2_)

        psi_bra = cplx.elementwise_mult(psi_bra1, psi_bra2)
        psi_bra_star = cplx.conjugate(psi_bra)
        return cplx.real(
            cplx.elementwise_division(psi_bra_star, psi_ket_star)
        )
```

Note the similarity in the implementation to that for the transverse magnetization observable from the last section, once the amplitude of a sample is substituted with the product of amplitudes drawn from each replica.

Using this observable, we can estimate the Rényi entropy of the region containing the first 5 sites in the chain using Eq. 5.10,

```
A = [0, 1, 2, 3, 4]
swap_ = SWAP(A)
swap_stats = swap_.statistics_from_samples(state, samples)
S_2 = -np.log(swap_stats["mean"])
```

We apply this measurement procedure to a TFIM chain with results shown in Fig. 5.3. As was the case with the magnetization observables, the trained RBM gives a good approximation to the second Rényi entropy for different subregion A sizes. Being a basis-independent observable, this constitutes a useful test on the ability of QuCumber to capture the full wavefunction from the information contained in a single-basis dataset for TFIM.

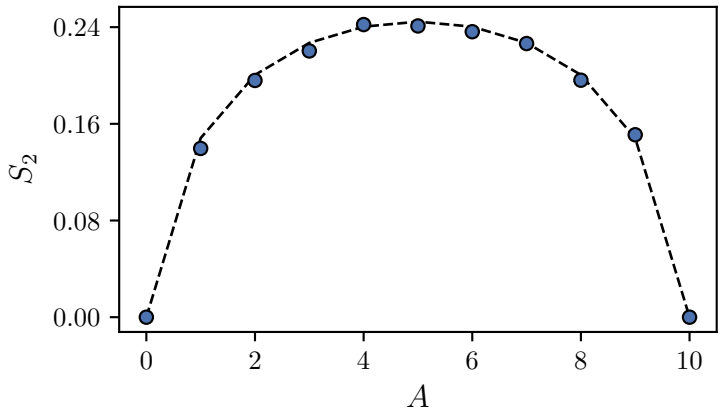


Figure 5.3: The second Rényi entropy for the TFIM chain with $N = 10$ spins. The number of sites in the entangled bipartition A is indicated by the horizontal axis. The markers indicate values obtained through the "Swap" operator applied to the samples from a trained RBM. The dashed line denotes the result from exact diagonalization.

5.4 Complex wavefunctions

For positive wavefunctions, the probability distribution underlying the outcomes of projective measurements in the reference basis contains all possible information about the unknown quantum state. However, in the more general case of a wavefunction with a non-trivial sign or phase structure, this is not the case. In this section, we consider a target quantum state where the wavefunction coefficients in the reference basis can be complex-valued, $\Psi(\boldsymbol{\sigma}) = \Phi(\boldsymbol{\sigma})e^{i\theta(\boldsymbol{\sigma})}$. We continue to choose the reference basis as $\boldsymbol{\sigma} = \boldsymbol{\sigma}^z$. We first need to generalize the RBM representation of the quantum state to capture a generic complex wavefunction. To this end, we introduce an additional RBM with marginalized distribution $p_{\boldsymbol{\mu}}(\boldsymbol{\sigma})$ parametrized by a new set of network weights and biases $\boldsymbol{\mu}$. We use this to define the quantum state as:

$$\psi_{\boldsymbol{\lambda}\boldsymbol{\mu}}(\boldsymbol{\sigma}) = \sqrt{p_{\boldsymbol{\lambda}}(\boldsymbol{\sigma})}e^{i\phi_{\boldsymbol{\mu}}(\boldsymbol{\sigma})/2} \tag{5.12}$$

where $\phi_{\boldsymbol{\mu}}(\boldsymbol{\sigma}) = \log(p_{\boldsymbol{\mu}}(\boldsymbol{\sigma}))$ (Torlai et al., 2018). In this case, the reconstruction requires a different type of measurement setting. It is easy to see that projective measurements in the reference basis do not convey any information on the phases $\theta(\boldsymbol{\sigma})$, since $P(\boldsymbol{\sigma}) = |\Psi(\boldsymbol{\sigma})|^2 = \Phi^2(\boldsymbol{\sigma})$.

The general strategy to learn a phase structure is to apply a unitary transformation $\boldsymbol{\mathcal{U}}$ to the state $|\Psi\rangle$ before the measurements, such that the resulting measurement distribution $P'(\boldsymbol{\sigma}) = |\Psi'(\boldsymbol{\sigma})|^2$ of the rotated state $\Psi'(\boldsymbol{\sigma}) = \langle\boldsymbol{\sigma}|\,\boldsymbol{\mathcal{U}}\,|\Psi\rangle$ contains fingerprints of the phases $\theta(\boldsymbol{\sigma})$ (Fig. 5.4). In general, different rotations must be independently applied to gain full information on the phase structure. We make the assumption of a tensor product structure of the rotations, $\boldsymbol{\mathcal{U}} = \bigotimes_{j=1}^{N}\hat{\mathcal{U}}_j$. This is equivalent to a local change of basis from $|\boldsymbol{\sigma}\rangle$ to $\{|\boldsymbol{\sigma}^{\boldsymbol{b}}\rangle = |\sigma_1^{b_1}, \ldots, \sigma_N^{b_N}\rangle\}$, where the vector $\boldsymbol{b}$ identifies the local basis b_j for each site j. The target wavefunction in the new basis is given by

$$\begin{aligned}\Psi(\boldsymbol{\sigma}^{\boldsymbol{b}}) = \langle\boldsymbol{\sigma}^{\boldsymbol{b}}|\Psi\rangle &= \sum_{\boldsymbol{\sigma}}\langle\boldsymbol{\sigma}^{\boldsymbol{b}}|\boldsymbol{\sigma}\rangle\langle\boldsymbol{\sigma}|\Psi\rangle \\ &= \sum_{\boldsymbol{\sigma}}\boldsymbol{\mathcal{U}}(\boldsymbol{\sigma}^{\boldsymbol{b}}, \boldsymbol{\sigma})\Psi(\boldsymbol{\sigma})\,,\end{aligned} \tag{5.13}$$

and the resulting measurement distribution is

$$P_{\boldsymbol{b}}(\boldsymbol{\sigma}^{\boldsymbol{b}}) = \left|\sum_{\boldsymbol{\sigma}}\boldsymbol{\mathcal{U}}(\boldsymbol{\sigma}^{\boldsymbol{b}}, \boldsymbol{\sigma})\Psi(\boldsymbol{\sigma})\right|^2. \tag{5.14}$$

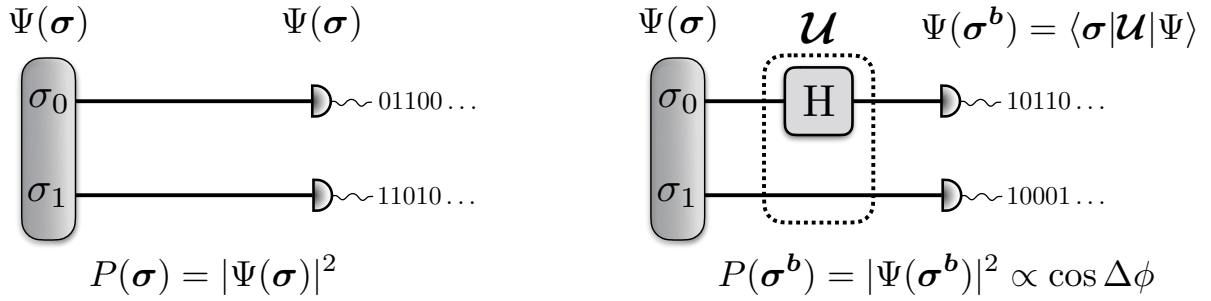


Figure 5.4: Unitary rotations for two qubits. (left) Measurements on the reference basis. (right) Measurement in the rotated basis. The unitary rotation (the Hadamard gate on qubit σ_0) is applied after state preparation and before the projective measurement.

To clarify the procedure, let us consider the simple example of a quantum state of two qubits:

$$|\Psi\rangle = \sum_{\sigma_0,\sigma_1} \Phi_{\sigma_0\sigma_1} e^{i\theta_{\sigma_0\sigma_1}} |\sigma_0\sigma_1\rangle \,, \tag{5.15}$$

and rotation $\boldsymbol{\mathcal{U}} = \hat{\mathrm{H}}_0 \otimes \hat{\mathcal{I}}_1$, where $\hat{\mathcal{I}}$ is the identity operator and

$$\hat{\mathrm{H}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5.16}$$

is called the *Hadamard gate*. This transformation is equivalent to rotating the qubit σ_0 from the reference σ_0^z basis the the σ_0^x basis. A straightforward calculation leads to the following probability distribution of the projective measurement in the new basis $|\sigma_0^x, \sigma_1\rangle$:

$$P_{\boldsymbol{b}}(\sigma_0^x, \sigma_1) = \frac{\Phi_{0\sigma_1}^2 + \Phi_{1\sigma_1}^2}{4} + \frac{1 - 2\sigma_0^x}{2} \Phi_{0\sigma_1}\Phi_{1\sigma_1} \cos(\Delta\theta) \,, \tag{5.17}$$

where $\Delta\theta = \theta_{0\sigma_1} - \theta_{1\sigma_1}$. Therefore, the statistics collected by measuring in this basis implicitly contains partial information on the phases. To obtain the full phases structure, additional transformations are required, one example being the rotation from the reference basis to the σ_j^y local basis, realized by the elementary gate

$$\hat{\mathrm{K}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -i \\ 1 & i \end{bmatrix} \,. \tag{5.18}$$

5.4.1 Setup

We now proceed to use QuCumber to reconstruct a complex-valued wavefunction. For simplicity, we restrict ourselves to two qubits and consider the general case of a quantum state with random amplitudes $\Phi_{\sigma_0\sigma_1}$ and random phases $\theta_{\sigma_0\sigma_1}$. This example is available in the online tutorial.[3] We begin by importing the required packages:

```
from qucumber.nn_states import ComplexWaveFunction
import qucumber.utils.unitaries as unitaries
import qucumber.utils.cplx as cplx
```

Since we are dealing with a complex wavefunction, we load the corresponding module `ComplexWaveFunction` to build the RBM quantum state $\psi_{\boldsymbol{\lambda\mu}}(\boldsymbol{\sigma})$. Furthermore, the following additional utility modules are required: the `utils.cplx` backend for complex algebra, and the `utils.unitaries` module which contains a set of elementary local rotations. By default, the set of unitaries include local rotations to the σ^x and σ^y basis, implemented by the $\hat{\mathrm{H}}$ and $\hat{\mathrm{K}}$ gates respectively.

We continue by loading the data[4] into QuCumber, which is done using the `load_data` function of the data utility:

```
train_path = "qubits_train.txt"
train_bases_path = "qubits_train_bases.txt"
psi_path = "qubits_psi.txt"
bases_path = "qubits_bases.txt"

train_samples, true_psi, train_bases, bases = data.load_data(
  train_path, psi_path, train_bases_path, bases_path)
```

As before, we may load the true target wavefunction from `qubits_psi.txt`, which can be used to calculate the fidelity and KL divergence. In contrast with the positive case, we now have measurements performed in different bases. Therefore, the training data consists of an array of qubits projections $(\sigma_0^{b_0}, \sigma_1^{b_1})$ in `qubits_train_samples.txt`, together with the corresponding bases (b_0, b_1) where the measurement was taken, in `qubits_train_bases.txt`. Finally, QuCumber loads the set of all the bases appearing the in training dataset, stored in `qubits_bases.txt`. This is required to properly configure the various elementary unitary

[3] The tutorial for complex wavefunctions can be found at https://qucumber.readthedocs.io/en/stable/_examples/Tutorial2_TrainComplexWaveFunction/tutorial_qubits.html

[4] The training dataset can be downloaded from https://github.com/PIQuIL/QuCumber/blob/master/examples/Tutorial2_TrainComplexWaveFunction/

rotations that need to be applied to the RBM state during the training. For this example, we generated measurements in the following bases:

$$(b_0, b_1) = (\mathrm{z}, \mathrm{z})\ ,\ (\mathrm{x}, \mathrm{z})\ ,\ (\mathrm{z}, \mathrm{x})\ ,\ (\mathrm{y}, \mathrm{z})\ ,\ (\mathrm{z}, \mathrm{y}) \tag{5.19}$$

Finally, before the training, we initialize the set of unitary rotations and create the RBM state object. In the case of the provided dataset, the unitaries are the $\hat{\mathrm{H}}$ and $\hat{\mathrm{K}}$ gates. The required dictionary can be created with `unitaries.create_dict()`. By default, when `unitaries.create_dict()` is called, it will contain the identity, the $\hat{\mathrm{H}}$ gate, and the $\hat{\mathrm{K}}$ gate, with the keys `Z`, `X`, and `Y`, respectively. It is possible to add additional gates by specifying them as

```
U = torch.tensor([[<re_part>], [<im_part>]], dtype=torch.double)
unitary_dict = unitaries.create_dict(<unitary_name>=U)
```

where `re_part`, `im_part`, and `unitary_name` are to be specified by the user.

We then initialize the complex RBM object with

```
state = ComplexWaveFunction(
   num_visible=2, num_hidden=2, unitary_dict=unitary_dict
)
```

The key difference between positive and complex wavefunction reconstruction is the requirement of additional measurements in different basis. Despite this, loading the data, initializing models, and training the RBMs are all very similar to the positive case, as we now discuss.

5.4.2 Training

Like in the case of a positive wavefunction, for the complex case QuCumber optimizes the network parameters to minimize the KL divergence between the data and the RBM distribution. When measuring in multiple bases, the optimization now runs over the set of parameters $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ and minimizes the sum of KL divergences between the data distribution $P(\boldsymbol{\sigma}^{\boldsymbol{b}})$ and the RBM distribution $|\psi_{\boldsymbol{\lambda\mu}}(\boldsymbol{\sigma}^{\boldsymbol{b}})|^2$ for each bases $\boldsymbol{b}$ appearing in the training dataset (Torlai et al., 2018). For example, if a given training sample is measured in the basis (x, z), QuCumber applies the appropriate unitary rotation $\boldsymbol{\mathcal{U}} = \hat{\mathrm{H}}_0 \otimes \hat{\mathcal{I}}_1$ to the RBM state before collecting the gradient signal.

Similar to the case of positive wavefunction, we generate the Hilbert space (to compute fidelity and KL divergence) and initialize the callbacks

```
state.space = nn_state.generate_hilbert_space(2)
callbacks = [
  MetricEvaluator(
    log_every,
    {"Fidelity": ts.fidelity, "KL": ts.KL},
    target_psi=true_psi,
    bases=bases,
    verbose=True,
    space=state.space,
  )
]
```

The training is carried out by calling the `fit` function of `ComplexWaveFunction`, given the set of hyperparameters

```
state.fit(
    train_samples,
    epochs=100,
    pos_batch_size=10,
    neg_batch_size=10,
    lr=0.05,
    k=5,
    input_bases=train_bases,
    callbacks=callbacks,
)
```

In Fig. 5.5 we show the total KL divergence and the fidelity with the true two-qubit state during training. After successfully training a QuCumber model, we can once again compute expectation values of physical observables, as discussed in Section 5.3.3.

5.5 Conclusion

We have introduced the open source software package QuCumber, a quantum calculator used for many-body eigenstate reconstruction. QuCumber is capable of taking input data representing projective measurements of a quantum wavefunction, and reconstructing the wavefunction using a restricted Boltzmann machine (RBM). Once properly trained, QuCumber can produce a new set of measurements, sampled stochastically from the RBM.

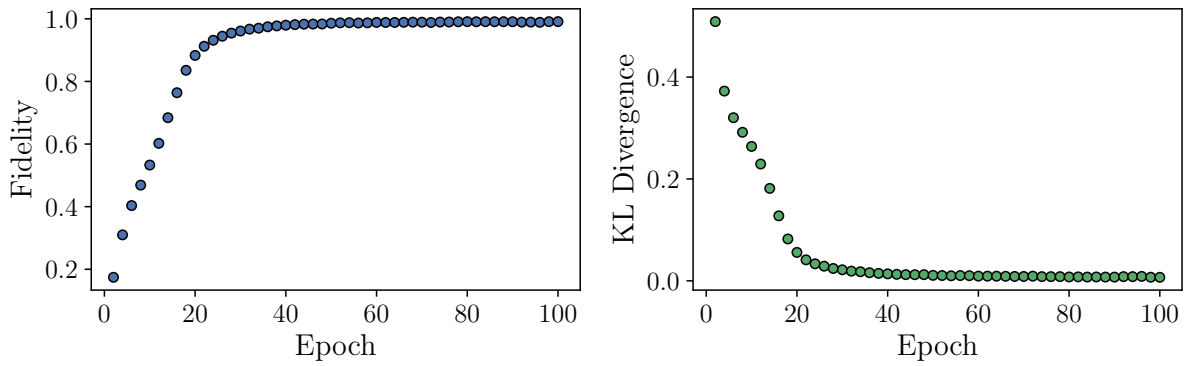


Figure 5.5: Training a complex RBM with QuCumber on random two-qubit data. We show the fidelity (left), and KL divergence (right), as a function of the training epochs.

These samples, generated in the reference basis, can be used to verify the training of the RBM against the original data set. More importantly, they can be used to calculate expectation values of many physical observables. In fact, any expectation value typically estimated by conventional Monte Carlo methods can be implemented as an estimator in QuCumber. Such estimators may be inaccessible in the reference basis, or they may be difficult to implement in the setup for which the original data was obtained. This is particularly relevant for experiments, where it is easy to imagine many possible observables that are inaccessible due to fundamental or technical challenges.

Future versions of QuCumber, as well as the next generation of quantum state reconstruction software, may explore different generative models, such as variational autoencoders, generative adversarial networks, or recurrent neural networks. The techniques described in this paper can also be extended to reconstruct *mixed* states, via the *purification* technique described by Torlai and Melko (2018). In addition, future techniques may include hybridization between machine learning and other well-established methods in computational quantum many-body physics, such as variational Monte Carlo and tensor networks (Carrasquilla et al., 2019).

Chapter 6

The scaling of RBM learnability of quantum states

The work presented in this chapter has been published as Sehayek et al. (2019).

The previous two chapters introduced Restricted Boltzmann Machines (RBMs) as generative models in ML (chapter 4) and demonstrated how RBMs can be implemented for the purpose of quantum state reconstruction (chapter 5) in physics. In this and the following chapter I present two reseach studies that use the QuCumber software to conduct experiments with RBMs deployed for modeling the distribution of the *transverse-field Ising model*.

As mentioned before, there are important differences between the experiments and approaches used to analyze DL systems – such as presented in chapter 3 – and the current line of research based on RBMs. In the beginning of chapter 4, I discussed two aspects: the *type of learning* and the NN model *architecture*. Specifically, I explained how generative modeling in unsupervised learning that we are deploying and studying in these works differs from supervised discriminative learning that underlies image classification tasks studied chapter in 3. Furthermore, I have discussed the important architecture difference between RBMs, which are shallow NN models, and deep NNs. Here, I would like to highlight another crucial difference: the dataset. In our study of RBMs as generative models for physical systems we use states of a quantum spin chain as training inputs. These spin configurations are obtained from a numerical simulation of the *transverse-field Ising model* in different coupling regimes. In contrast to the natural images used in computer vision datasets, the spin states are less complex in their representation: Firstly, each spin state is a one-dimensional binary sequence of $+1$ and -1, with its length ranging from 10 to 100

(as we consider relatively small system sizes), whereas each natural image is a set of three two-dimensional arrays of real numbers representing pixel values on RGB scale, with dimensions ranging between 28−by−28 to 1024−by−1024. Naturally, datasets with smaller input dimension require NN models of correspondingly smaller size and repersentational power. However, this scaling argument applies only on a coarse level; as we show in the following two studies, the expressive power and thus the size of the NN model scales as a function of the intrinsic dataset complexity which defines the difficulty of the learning task. More specificially, in the case of a quantum spin model this "intrinsic complexity" is primarily determined by the interaction regime or the quantum phase. For instance, deep in the disordered paramagnetic phase we expect to find spin states with a relatively simple structure because of the absence of spin-spin correlations, while at the quantum critical point we expect the state structure to be most complex.

In general, the following two studies aim at characterizing RBM-based learning for physical systems and investigate the capacity and the experessive power of RBMs deployed for the task of quantum state reconstruction. In these projects, we leverage the advantage of working with a dataset that is derived from a physical system, as it provides us with a variety of model quality measures and thus allows us to scrutinize RBM learning a level that is not accessible in other settings. In this way, we flip the roles of ML and physics and use our knowledge of physical systems to gain insight into ML systems.

6.1 Abstract

Generative modeling with machine learning has provided a new perspective on the data-driven task of reconstructing quantum states from a set of qubit measurements. As increasingly large experimental quantum devices are built in laboratories, the question of how these machine learning techniques scale with the number of qubits is becoming crucial. We empirically study the scaling of RBMs applied to reconstruct ground-state wavefunctions of the one-dimensional transverse-field Ising model from simulated projective-measurement data. We define a learning criterion via a threshold on the relative error in the energy estimator of the machine. With this criterion, we observe that the number of RBM weights required for accurate representation of the ground state in the most complex case – near criticality – scales quadratically with the number of qubits. By pruning small parameters of the trained model, we find that the number of weights can be significantly reduced while still preserving an accurate energy expectation value. However, a systematic study of pruning presented in the following chapter reveals that the fulfillment of the ROE criterion is not sufficient to ensure that the model has retained an accurate state reconstruction.

6.2 Motivation

Generative models are a powerful class of machine learning algorithms that seek to reconstruct an unknown probability distribution $p(\boldsymbol{x})$ from a set of data $\boldsymbol{x}$. After training, generative models can be used to estimate the likelihood of new data not contained in the original set, or to produce new data samples for various purposes. Recently, industry-standard generative models have been repurposed by the physics community with the goal of reconstructing a quantum wavefunction from projective measurement data. While several generative modeling techniques are available for quantum state reconstruction, by far the most well-studied involves restricted Boltzmann machines (RBMs) (Torlai and Melko, 2016; Torlai et al., 2018; Chen et al., 2018; Cheng et al., 2018; Carleo et al., 2018b; Carrasquilla et al., 2019). RBMs can be used to explicitly parametrize a probability distribution $p(\boldsymbol{x})$, and, through a suitable complex generalization, a quantum wavefunction (Torlai et al., 2018; Carleo and Troyer, 2017b). One main application of RBMs is the data-driven reconstruction of experimental states, which has recently been demonstrated for a Rydberg-atom quantum simulator (Torlai et al., 2019). These and other uses have been covered extensively in the literature, including several recent reviews (Torlai and Melko, 2020; Melko et al., 2019; Carleo et al., 2019).

With the steady increase in the size of experimental quantum devices, the question of how data-driven quantum state reconstruction scales with the number of qubits is of paramount importance. While many results have been reported for fixed finite-size reconstructions, less work has been done in the way of scaling analyses (Czischek et al., 2019). Particularly important is the difference in scaling complexity of approximate machine learning methods for practical reconstructions, as compared to full quantum-state tomography that in general scales exponentially (Paris and Rehacek, 2004).

Here, we present a systematic study of the scaling of the computational resources required for accurate reconstruction of a quantum state. In particular, we focus on RBMs used to reconstruct the ground-state wavefunction of a one-dimensional transverse-field Ising model, which has a positive-real representation. Our training data is a set of projective measurements sampled independently from a simulated tensor-network wavefunction. We define a learning criterion based on the accuracy of the energy estimator of the RBM. The state reconstruction is considered successful when the relative error of the energy estimator is smaller than a fixed threshold. We target in particular two contributions to the asymptotic scaling behavior in the many-qubit limit: the representational power of the neural network, i.e., the *expressivity* of the parametrization of the state, and the amount of data required to train the model, also known as the *sample complexity*.

In the present work, we use the QuCumber [99] software package to implement and

train a positive-real RBM.

6.3 Defining a scaling study

6.3.1 Physical system and RBM setup

We are interested in probing the asymptotic scaling of the computational resources required to reconstruct a quantum state using an RBM. The training set comprises projective measurement data produced from the ground-state wavefunction of the one-dimensional transverse-field Ising model (TFIM) defined by the Hamiltonian

$$H = -J \sum_{\langle ij \rangle} \sigma_i^z \sigma_j^z - h \sum_i \sigma_i^x, \tag{6.1}$$

where $\sigma^{x,y,z}$ are Pauli operators, defined over N sites (or qubits), and $\langle ij \rangle$ denotes nearest-neighbor pairs on a one-dimensional lattice with open boundary conditions. This model is thoroughly studied in the condensed matter and quantum information literature, and serves as a standard benchmark for many numerical methods, such as Quantum Monte Carlo (QMC) (Sandvik, 2003; Inack et al., 2018), Tensor Networks (TNs) (Vidal, 2007), or more recent quantum optimization algorithms (Ho and Hsieh, 2019; Ho et al., 2019; Beach et al., 2019b). We generate training data from a density matrix renormalization group (DMRG) simulation (Ferris and Vidal, 2012) for various values of h/J using the ITensor library (Fishman et al., 2020). The measurements of the ground-state wavefunction are produced in the σ^z basis.

The Perron-Frobenius theorem guarantees that when the Hamiltonian Eq. (6.1) has negative off-diagonal matrix elements in the σ^z (computational) basis, the ground-state wavefunction is positive-real. Thus, there is a direct mapping between the wavefunction and a probability distribution, $\psi(\boldsymbol{\sigma}) = \sqrt{p(\boldsymbol{\sigma})}$. This allows for a significant simplification in the RBM network structure, since complex phases or signs need not be parametrized. In addition, the computational basis is trivially informationally complete, enabling training from data produced only in the σ^z basis (Torlai et al., 2018).

6.3.2 Learning criterion

In order to quantify the resources required for the data-driven reconstruction of the ground-state wavefunction for the TFIM, one must be able to assess when the learning is "complete". Generally, the fidelity is considered a standard measure of the closeness of two

quantum states, such as a target state and an approximate reconstructed state. However, in more generic situations than ours, where a TN representation of the target quantum state may not be available, calculations of the fidelity typically scale exponentially, which renders them intractable for even moderate numbers of qubits. An alternative method for defining the accuracy of a reconstruction is to measure expectation values of local observables. Such expectation values can be efficiently calculated through standard estimators from samples produced by the RBM. Importantly, these can be compared with the exact values measured from our DMRG simulations, or through other methods such as Quantum Monte Carlo (QMC) that do not admit an explicit representation of the ground state.

The relative error between an RBM estimator and an exact DMRG expectation value will be referred to as the *relative observable error* (ROE). For the current study, we define the learning criterion through the ROE in the expectation value of the energy, which can be calculated from the RBM using standard Markov Chain Monte Carlo techniques. Take $\bar{U} = \langle H \rangle_{\mathrm{RBM}}$ to be the average of the energy estimator calculated from n samples generated by the RBM. Since n is finite, a statistical error exists in the estimator, quantified by the standard deviation σ. To account for this in a relative error measure, we compute the Gaussian confidence interval given by $\bar{U} \pm C\frac{\sigma}{\sqrt{n}}$. The value of $C = 2.576$ corresponding to 99% confidence will be used throughout this study. If $U = \langle H \rangle_{\mathrm{exact}}$ is the exact value of the energy estimator (calculated, e.g., with DMRG), then we can upper-bound the ROE by the larger relative error value of the confidence interval:

$$\epsilon = \max \left| \frac{U - (\bar{U} \pm C\sigma\sqrt{n})}{U} \right| . \tag{6.2}$$

Essentially, this means that we consider the learning to be "complete" when our desired upper bound on the ROE is satisfied 99% of the time on our sample size. We find empirically that $\epsilon = 0.002$ is a reasonable value that can be achieved by RBMs trained on TFIM data with conventional algorithms for $N \leq 100$ qubits. At smaller values (e.g., $\epsilon = 0.001$) training becomes impractical for $N > 50$, while for larger values we observed that the results reported below remain qualitatively the same; thus, we choose to use $\epsilon = 0.002$ in this study.

With this learning criterion, we analyze the scaling behavior of the RBM by controlling two variables: the number of model parameters per qubit and the number of training measurements M, i.e., the sample complexity. However, we note that, as typical in machine learning studies, many other variables exist that are related to network architecture, learning rates, batch size, etc. – referred to as *hyper-parameters*. We set these hyper-parameters consistently for all values of h/J and all system sizes N.

6.4 Results

We present numerical results for the scaling of computational resources for reconstruction of the TFIM ground-state wavefunction for several values of h/J. In order to systematically investigate scaling, we control variables of interest in separate experiments as described in the following sections.

6.4.1 Scaling of the model parameters

To begin, we are interested in the minimal number of RBM parameters per qubit required to faithfully reproduce the ground-state energy. We parametrize this with the scaling of the size of the hidden layer, N_h. We consider the critical point, corresponding to $h/J = 1$, as well as the ferromagnetic and paramagnetic phases. For each value of N, we produce large numbers of projective measurements of σ^z values using the DMRG simulation of the TFIM. Then, assuming that effectively the number of available training samples $M \to \infty$, we increase the number of hidden units N_h until the learning criterion is satisfied for each value of N.

Our procedure is illustrated in Figure 6.1 for a fixed system size of $N = 50$. In the main plot, corresponding to $h/J = 1$, we observe that the specified learning criterion $\epsilon = 0.002$ can not be achieved for $N_h < 25$. The minimum number of parameters required to accurately represent the ground-state wavefunction is thus $N_h = 25$. The inset illustrates the dependence of N_h on field values near $h/J = 1$, where the quantum wavefunction is most entangled. One would expect that in the limit $N \to \infty$ the number of parameters required to accurately parametrize the wavefunction would be maximal at $h/J = 1$. Curiously, we find that this peak occurs around $h/J \approx 0.8$, slightly on the ferromagnetic side from the critical point. We hypothesize that this feature might be tied to the magnetization of the underlying dataset used for training, which was produced by our DMRG simulations in ITensor. For the maximum bond dimension that we employ (2000), the expected $\mathbb{Z}_2$ symmetry is not realized below certain values of the transverse field strength h when the number of qubits is large. Furthermore, a similar phenomenon has been observed previously in studies of the relative energy in diffusion Monte Carlo (Inack et al., 2018) and a recent variational imaginary time ansatz (Beach et al., 2019b). It would be an interesting topic of future study.

The result of repeating the above procedure for various numbers of qubits N is illustrated in Figure 6.2. For values of h/J deep within the ferromagnetic or the paramagnetic

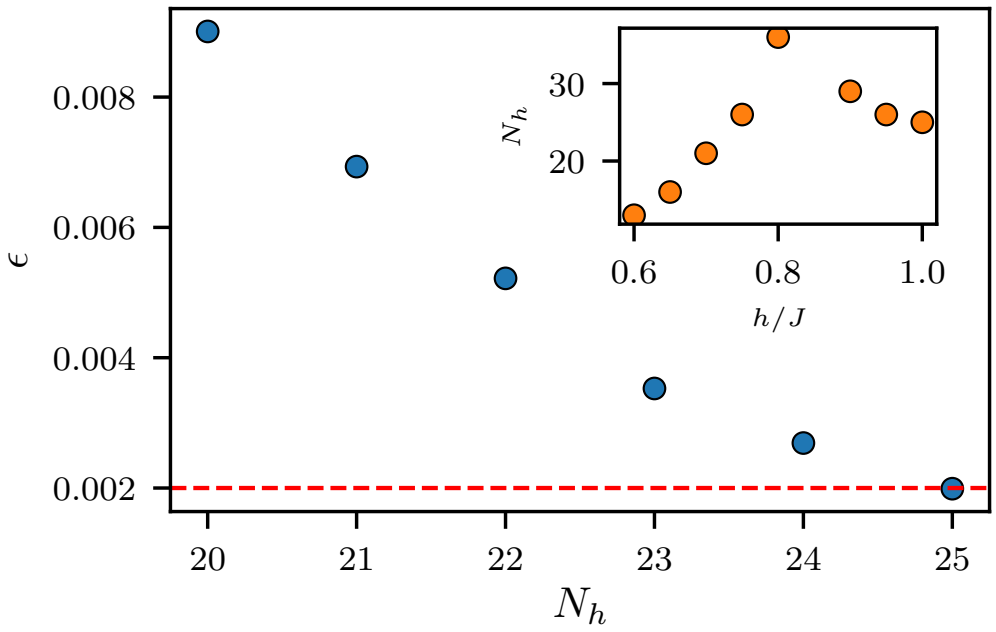


Figure 6.1: The procedure used to determine the RBM expressiveness required to represent the TFIM wavefunction at $h/J = 1$ with $N = 50$ qubits. The number of hidden units N_h is increased until the desired ϵ is achieved. The inset illustrates the number of hidden units required for convergence to $\epsilon \leq 0.002$ for different values of h/J near criticality. The position of the peak is discussed in the main text.

phase, the required minimum number of hidden units scales as $N_h \sim \mathcal{O}(1)$ in the asymptotic limit of large N. This reflects the informational simplicity of the dataset close to the ferromagnetic or paramagnetic limits. Near $h/J \approx 1$, the scaling is clearly linear, $N_h \sim N$, meaning that the leading-order scaling of the number of parameters is $\mathcal{O}(N^2)$, as each additional hidden unit quadratically scales the number of elements in the weight matrix $\boldsymbol{W}$. Due to the presence of the bias terms in the RBM energy function (Eq. 4.1), we would also expect a sub-leading term that scales proportionally to N; however, as noted in the Appendix, for data sets with an underlying $\mathbb{Z}_2$ symmetry, these bias terms do not represent independent parameters for the purpose of wavefunction reconstruction. Finally, we note that for larger ROE thresholds $\epsilon > 0.002$ the prefactors and slopes are different, but the asymptotic scaling of the number of hidden units still remains linear near criticality.

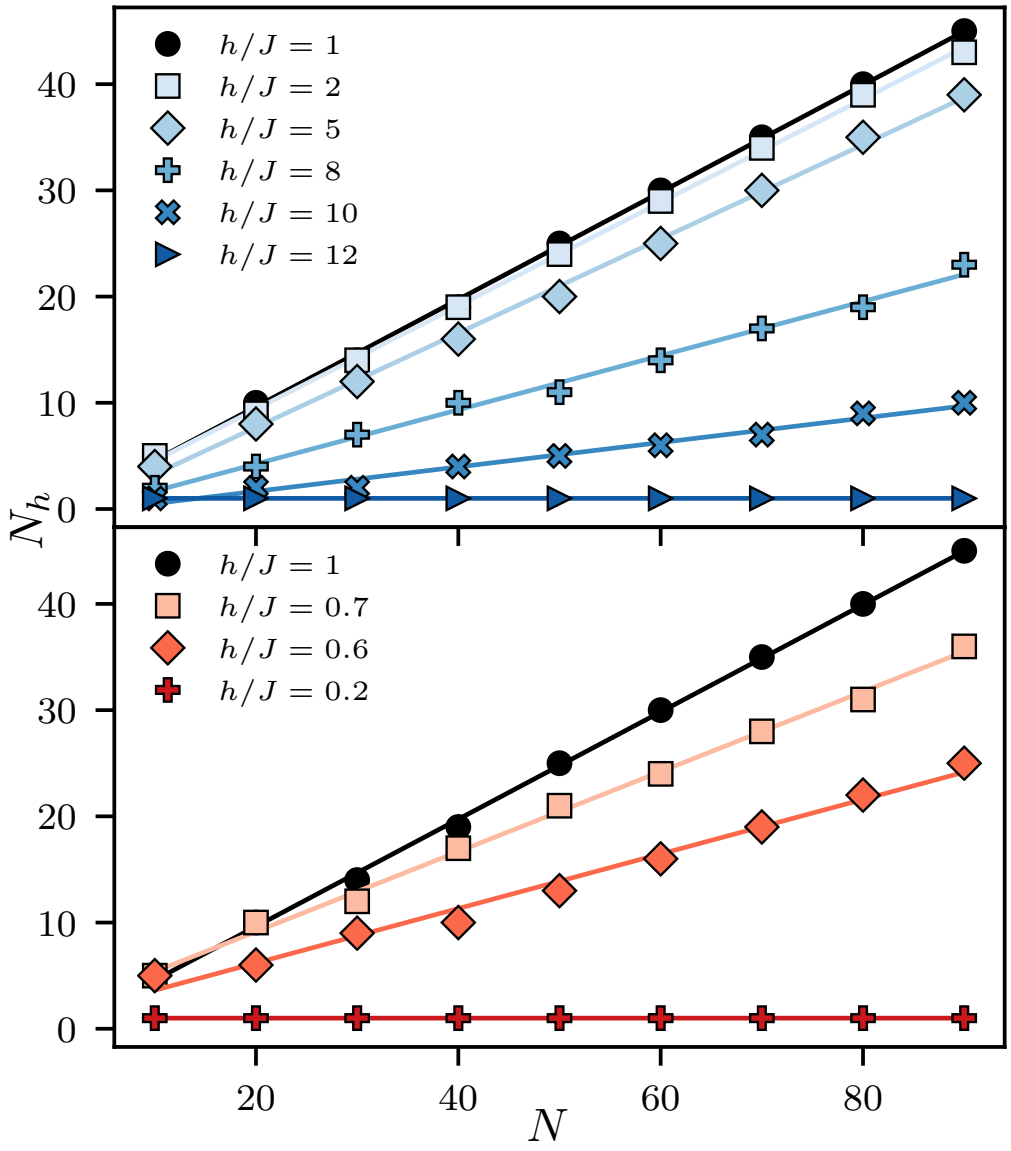


Figure 6.2: Minimum number of hidden units N_h required for $\epsilon \leq 0.002$ for various values of h/J. Straight lines are fits to the data.

Comparison to MPS

We draw a brief comparison between the above results and naive expectations for the scaling of the number of parameters in the simplest TN representation of a one-dimensional wavefunction – the matrix product state (MPS). We refer the reader to recent reviews on the topic, e.g., Orús (2014, 2019). For a finite-size MPS with N matrices, the simplest estimate for the number of parameters required to store a wavefunction is $\mathcal{O}(N\chi^2)$, where χ is the *bond dimension*. The bond dimension required for good representation accuracy depends on the amount of entanglement in the system. In the presence of a bipartition, the entanglement entropy S is upper-bounded by the logarithm of the rank of the reduced density matrix. For an MPS, every bond that lies on the bipartition therefore gives an entropy contribution of at most $\log(\chi)$, leading to the scaling rule of $\chi \sim \exp(S)$. For the one-dimensional TFIM, we expect $S \sim \mathcal{O}(1)$ in the ordered phases, and $S \sim \log(N)$ at the quantum critical point. This yields a naive scaling of $\mathcal{O}(N)$ in the ordered case, and $\mathcal{O}(N^3)$ at the critical point. We note that our RBM scaling is more consistent with a translationally-invariant MPS encoding, where these asymptotic scaling complexities are both reduced by a factor of N. This reduction is expected in long one-dimensional chains, where tensors sufficiently far from the edges are typically identical.

6.4.2 Scaling of sample complexity

Above, we studied the minimum number of RBM parameters required to find an accurate ground-state energy, assuming access to an infinite amount of training data. We now determine the minimal sample complexity required to accurately train this number of parameters. We focus on $h/J = 1$, and fix the ratio $\alpha = N_h/N$ for several values near $1/2$. Then, repeating the procedure from the last section, we increment the number of training examples M by 2500 until the ROE learning criterion $\epsilon \leq 0.002$ on the RBM energy estimator is achieved. This procedure is repeated for a number of different initial weight configurations, and the results are averaged. The resulting scaling of the sample complexity is shown in Figure 6.3.

The results suggest that for N_h/N near $1/2$, the sample complexity scales linearly in the number of qubits. Combining the asymptotic scaling results from the previous two sections, $N_h \sim N$ and $M \sim N$, suggest that the number of samples per parameter required to train a minimally-expressive machine scales as $N/N^2 \sim 1/N$. In other words, the relative "data cost" required to train a new weight parameter decreases with an increasing number of qubits. A linear scaling of the sample complexity was also observed in a recent

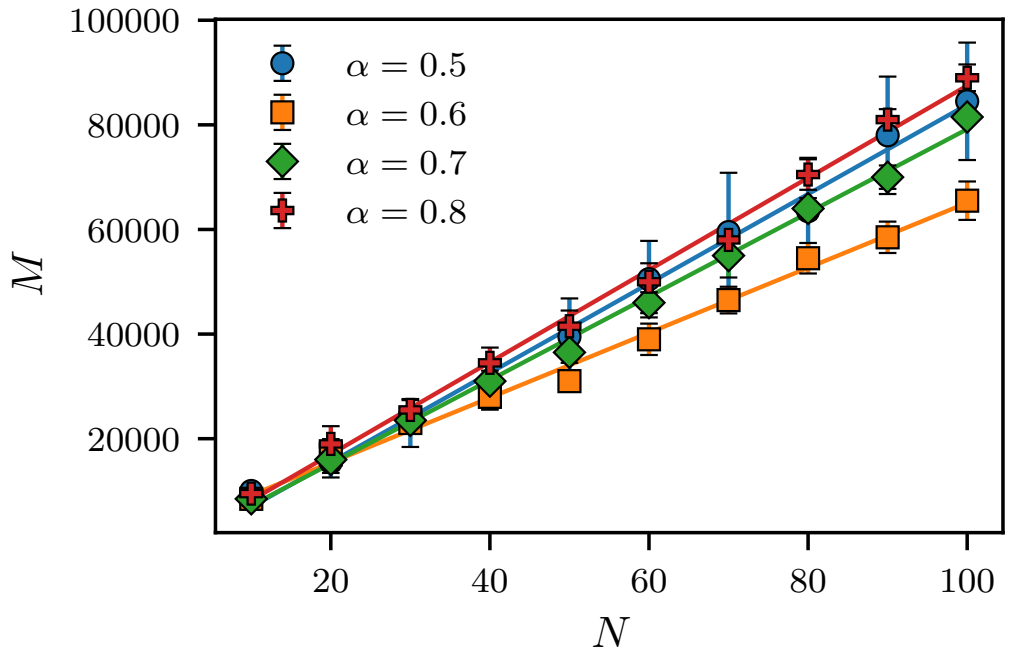


Figure 6.3: The minimum number of training examples M required for $\epsilon \leq 0.002$ for the TFIM at the critical point $h/J = 1$ for different ratios of the number of hidden to visible units $\alpha = N_h/N$.

generative modeling scheme based on positive operator-valued measurements (Carrasquilla et al., 2019).

We remark that a sample complexity linear in N is consistent with observations on the PAC-learnability of quantum states. Aaronson (2007) argues that, if one is only concerned about learning a state well enough to predict the outcomes of most measurements drawn from it, the exponential cost usually associated with full state tomography is reduced to a linear scaling in N. This is what we find in Figure 6.3. Indeed, a characteristic of the Aaronson learning theorem is the assumption that the training samples are drawn *independently* from the probability distribution. This is exactly the setting that we employ in training the RBM in the present work. Hence it is reasonable to expect the theorem to apply.

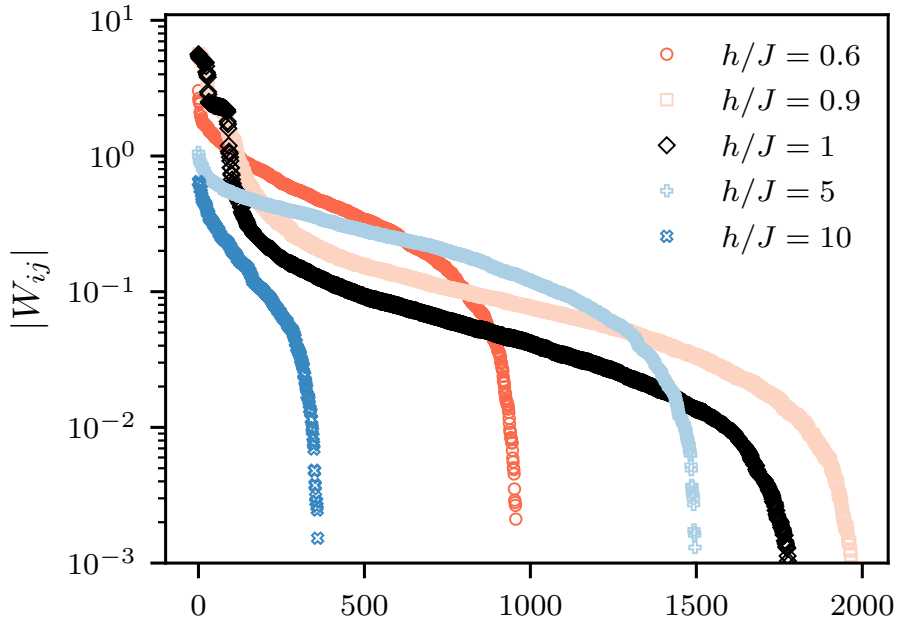


Figure 6.4: Weight magnitudes, sorted in descending order from left to right, for various transverse field values and $N = 60$. Converged RBM models from the parameter study shown in Figure 6.2 are used here.

6.4.3 Reducing the number of model parameters post-training

Further insight on the RBM result can be obtained by examining the distribution of the absolute weight values $|W_{ij}|$ in a typical trained model. Figure 6.4 shows the magnitude of each individual weight, sorted in decreasing order from left to right, on a logarithmic scale. Near criticality, the largest contribution is given by the first $10 - 20\%$ of weights; then the weight values decrease exponentially in magnitude, eventually falling off even more rapidly.

In the previous section, we found that the minimal number of hidden units required to satisfy our chosen ROE scales approximately as $N_h \approx \frac{1}{2}N$ near the TFIM quantum critical point. Implicit in this result is the RBM optimization procedure used to train the machine: a stochastic gradient descent that minimizes the KL-divergence (Torlai and Melko, 2020). Since it is not obvious that the scaling behavior is independent of this optimization procedure, it is fair to ask the question: Is it possible to find more efficient representations – with fewer model parameters – by modifying the learning protocol? Indeed, it is known that the required number of model parameters is intertwined with the specifics of the training procedure. In particular, it has been found that the over-parametrization inherent to

N	10	20	30	40
original	50	200	420	760
pruned	20	50	79	119

Table 6.1: The number of weights required to achieve $\epsilon \leq 0.002$ at the critical point $h/J = 1$. Results for the "original" RBM are taken from Figure 6.2.

deep neural networks can ease and accelerate their optimization by (stochastic) gradient descent (Lopez-Paz and Sagun, 2018; Livni et al., 2014; Arora et al., 2018; Allen-Zhu et al., 2018; Sankararaman et al., 2019).

Figure 6.4 offers a clue that the RBM parametrization may not be optimal (i.e., minimal) for the final trained wavefunction by demonstrating that the distribution of the weight magnitudes in a trained model is very non-uniform: 10 to 20% of the weights have values that are orders of magnitude larger than the rest. Recent machine learning literature has studied the relative importance of these smaller weights with a procedure called *pruning*. Following the ideas of Han et al. (2015a) and Mocanu et al. (2016), we define a pruning procedure for our scaling study in the following steps:

1. Start from the original, converged trained model (e.g., Figure 6.4, with $N_h = \frac{1}{2}N$ for $h/J = 1$).

2. Set a threshold δ for the weight magnitudes. If a given $|W_{ij}| < \delta$, set $W_{ij} = 0$, and freeze it for the following steps.

3. Fine-tune the pruned model by running several more training iterations until the desired accuracy (as defined by the ROE learning criterion) is restored.

4. Repeat steps 2-4, pruning additional weights until the model fails to fulfill the learning criterion.

We choose the pruning threshold such that 40% of the non-zero weights are pruned in the first iteration, and 5% of the non-zero weights in each following iteration. Note also that in this procedure we do not prune biases.

We apply weight pruning to our trained RBM focusing on the critical point of the TFIM, and find that a significant reduction in the number of RBM parameters required to correctly capture the critical TFIM ground-state energy can be achieved for all system sizes. The results for several small numbers of qubits are presented in Table 6.1. We interpret this to mean that the standard training of an RBM with contrastive divergence benefits from

an over-parameterization, employing more weights than is strictly required for accurate expression of the TFIM wavefunction in order to make the optimization more navigable. We note that in some rare cases, pruning a very small number of weights seriously alters the ROE, highlighting that some paths through the optimization landscape may depend on weight parameters that are not redundant. For this reason, rigorous uncertainty intervals on our results are difficult to estimate at present.

The success of the pruning procedure opens up the possibility of systematically searching for a change in scaling behavior. This analysis was conducted in a separate study and is presented in the next chapter.

6.5 Summary

We have empirically studied the scaling of computational resources required for the accurate reconstruction of positive-real wavefunctions using generative modeling with an RBM. We obtained scaling results by examining the energy estimator calculated from an RBM after training on simulated projective-measurement data for the ground state of a one-dimensional TFIM with open boundary conditions.

We have found that deep within the ferromagnetic and paramagnetic phases, the number of RBM parameters required for accurate representation of the ground state is $\mathcal{O}(1)$. As the transverse field is varied to approach the quantum critical point between these two phases, the state becomes more challenging to reconstruct, as expected due to long-range quantum correlations that arise there. At the critical point, we observe that under standard RBM training procedures the number of parameters grows quadratically in the number of qubits, $\mathcal{O}(N^2)$. The minimum number of measurements required to train this number of parameters scales linearly with the number of qubits, $\mathcal{O}(N)$. Interestingly, we find that the number of parameters required for an accurate reconstruction – with the accuracy measure based on the energy ROE – can be significantly reduced post-training by pruning small weights and fine-tuning the RBM by a small number of additional training iterations.

6.6 Discussion

In the current study, an RBM reconstruction of the ground-state wavefunction was defined to be "accurate" when the relative error between the RBM estimator and the exact energy value was below a fixed threshold. Thus, our scaling results are subject to the caveat

that they could change if other criteria were to be considered, such as the convergence of fidelity or correlation functions. We dedicate a separate study to this question, which is presented in the next chapter. In the present case, convergence of the relative error in the energy produces several interesting results: First, for a standard optimization procedure with contrastive divergence, the number of weights required for accurate reconstruction is at best constant (deep in the ferromagnetic/paramagnetic phases). At worst, this scaling is quadratic; this occurs near the quantum critical point between the two phases. It is interesting to note that such scaling is consistent with that expected from a translationally-invariant matrix product state encoding. In addition, the minimum number of samples required to converge the energy at the critical point is observed to scale linearly with the number of qubits. This is consistent with a theorem by Aaronson (2007) that predicts a linear scaling in a similar setting for PAC-learning.

Further, we present evidence that the number of parameters required to represent the ground state is drastically affected by the RBM learning procedure. By employing a pruning technique that sets small weights to zero, then fine-tuning the remaining model parameters through additional training, we observe a very significant reduction in the number of parameters required to accurately reproduce the energy. However, we have not examined how other observables are affected by pruning, and thus not verified that the reconstruction remains faithful. We address this shortcoming with a thorough study presented in the next chapter.

Indeed, numerous recent results have highlighted the benefit that overparametrization provides for optimizing deep learning models (Allen-Zhu et al., 2018; Sankararaman et al., 2019; Lopez-Paz and Sagun, 2018). In this work, we have discovered that RBMs trained on simulated measurement data for positive-real wavefunctions may as well be aided by over-parametrization – beyond what is needed for the theoretical representation of the quantum state – as a means of assisting the standard optimization procedure of minimizing the KL divergence via contrastive divergence. The question of how to systematically mitigate this overparametrization while still maintaining the ease of optimization is an active area of research (Frankle and Carbin, 2018; Zhou et al., 2019; Lee et al., 2018b), one whose successes will be of great use for more efficiently representing and studying quantum systems. A principled pruning approach could provide a systematic way of searching for the minimal model expressivity required for a given quantum state. It would then be interesting to compare the obtained results to theoretical expectations for the representational capacity of RBMs required for quantum ground-state wavefunctions (Chen et al., 2018; Gao and Duan, 2017b; Glasser et al., 2018b).

It is natural to wonder what the scaling of computational resources is for reconstructing quantum states that are not real or positive. To this end, the present results point towards

a rich field of similar scaling studies that should be pursued on a variety of quantum many-body models in the future. The question of scaling is also especially pertinent for state-of-the-art experiments, such as fermionic quantum simulators (Mazurenko et al., 2017), wavefunctions generated by quantum dynamics (Lanyon et al., 2017; Keesling et al., 2018), or quantum chemistry calculations with superconducting circuits (Kandala et al., 2017). In contrast to positive wavefunctions, the reconstruction for a general quantum state, with a suitably modified RBM, demands training data from an extended set of measurement bases. The ability to theoretically identify the minimal set, and how the size of this set scales with the number of qubits, will ultimately determine the feasibility of integrating this type of machine learning technology into such near-term quantum devices.

In conclusion, we have proposed a systematic procedure to evaluate the scaling of resources for reconstructing positive-real wavefunctions with RBMs. A tighter threshold in the reconstructed energy accuracy, or improved neural-network parametrization of non-positive states, will likely require a more powerful breed of generative model. Recurrent neural networks, transformers, and other autoregressive models are currently being considered in this context. In light of the fact that current intermediate-scale quantum devices are already capable of producing training data on tens and even hundreds of qubits, we expect these and similar scaling studies will be pursued in earnest in the near future.

Chapter 7

Pruning the RBM

In the previous chapter we studied how a Restricted Boltzmann Machine (RBM) employed as a generative model for the task of quantum state reconstruction scales with the number of qubits. Preliminary results obtained in that study indicated that the *quadratic* scaling between the number of qubits and the number of weights in the RBM changes to *linear* when magnitude-based pruning is applied post training. In this follow-up study we investigate the effects of pruning in detail. We find that while pruned models still fulfill the energy ROE criterion, many of their other properties deteriorate significantly. Our results highlight the fact that model size reduction through weight pruning does not come for free, and stress the importance of monitoring *all* accessible model properties if applying pruning.

7.1 Motivation

Pruning is a common technique employed in machine learning (ML) with the goal of reducing the overall number of parameters in a neural network (NN). The fact that pruning can substantially reduce the the number of weights without degrading the performance of a NN has been known since the early days of ML (Reed, 1993), but the real explosion of interest in this technique happened only in recent years when deep NNs (DNNs) were scaled up to handle large datasets and industrial applications. DNNs – particularly the ones that perform best – require enormous amounts of computation and memory, and thus the matter of compressing them became of immediate concern. The most successful approach is *iterative magnitude-based pruning post training*, which eliminates the smallest weights from a trained NN stepwise and allows the remaining parameters to adjust by doing a couple training iterations after each pruning step (Thimm and Fiesler, 1995; Ström, 1997; Han

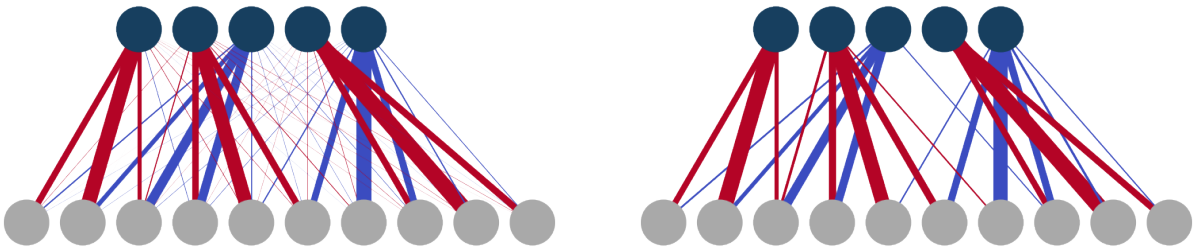
et al., 2016). This method can substantially reduce the computational cost of inference and enable deployment of NN-based applications in resource-constrained environments, such as mobile devices (Han et al., 2015b; Yang et al., 2017; Sze et al., 2017). However, pruning *post training* means that the NN still has to be trained at its full size. A variety of newer works propose pruning based on alternative criteria (Lee et al., 2018b; Evci et al., 2019; Wang et al., 2020), but to date there is no successful approach to pruning *before training* (Frankle et al., 2020). Moreover, the principles of why pruning works in general are not understood and it is unclear what determines a successful sparsity pattern for a given NN. For the most recent review of the current state of pruning research in ML see Blalock et al. (2020).

As all fields involving "big data", physics would benefit from more efficient DNNs. In particular in quantum many-body physics, we are ultimately interested in modeling large physical systems, and the anticipated power of ML techniques lies in their ability to handle system sizes that conventional numerical methods can not. One of the most important applications requiring NNs with as few parameters as possible is the reconstruction of a quantum many-body state from data. In the simplest case of a quantum state prepared on a number of qubits, the task involves learning the probability distribution underlying a set of projective measurements given by the Born rule. This is most immediately accomplished by a *generative model*, whose parameters are trained to maximize the likelihood that it accurately represents the data distribution. In chapter 5 we discussed this task and the implementation of a generative model based on a Restricted Boltzmann Machine.

Perhaps more importantly, physics also has the potential to advance the theoretical understanding of pruning and sparsity in NNs. The main advantage is that physics problems offer a practical test bed for ML methods like no other dataset, because they are well-characterized: For instance, for the quantum state reconstruction problem we can describe the entire state space, generate input data, and we know the characteristics of the probability distribution. This knowledge opens up more ways to evaluate the learning procedure and results, and thus to obtain insights into the processes of RBM-learning and pruning. Additionally, we can leverage our expertise from other computational methods, such as for example Tensor Network techniques used for quantum state representation which have proven scaling and compression limits (Chen et al., 2018).

7.2 Introduction

Due to their foundations in the Ising model of statistical mechanics, RBMs have become familiar to condensed matter and quantum information physicists. In quantum physics,



(a) before pruning (b) after pruning

Figure 7.1: Graphs of a trained RBM before and after pruning. Lines represent weights, line thickness corresponds to weight magnitude, and color indicates the sign (red for +, blue for −).

RBMs have been used for both data-driven state reconstruction, and as a variational ansatz where parameters are optimized with knowledge of the Hamiltonian. The combination of high representational capacity and a proven training heuristic makes RBMs compelling candidates for data-driven wavefunction reconstructions. The full connectivity of the RBM provides an upper bound on the entanglement that it can represent in a physical wavefunction. This upper bound – corresponding to the "volume law" – is sufficiently expressive to capture the entanglement behavior of any known class of quantum many-body states. However, ground states of local Hamiltonians are expected to obey *sub*-volume-law scaling, which raises the question whether RBMs with sparser connectivity can be found, and would lead to more efficient reconstruction schemes.

The idea of pruning comes naturally when we examine the weights in a trained RBM. Consider Figure 7.1a which shows the graph of a fully trained RBM used to reconstruct the state of a 10-qubit TFIM. It is apparent that only a small number of weights have large magnitudes, and all other weights are much smaller in comparison. This suggests that the small-magnitude weights might be not significant, and can thus be set to zero (*pruned*) without affecting model performance. Figure 7.1b shows the RBM after pruning, which still performs adequately according to *some* metrics. However, as we show in detail in the following, pruning affects various model qualities in different proportions.

We have touched upon this subject in the previous chapter, which focused primarily on the scaling of RBM model size and dataset size with the number of qubits in the 1d TFIM model. As our main focus was RBM training, we used energy ROE as the sole criterion for model evaluation, which will turn out to be an insufficient metric in case of pruning. In the study presented here, we train RBMs on data for various system sizes and transverse field strengths h/J, and subsequently apply pruning. In addition to the standard loss function,

the KL divergence, we examine the effect of pruning on physical properties, namely the fidelity, energy, order parameter and 2-point correlation function.

7.3 Methods

Our setup is the same as in the scaling study presented in chapter 6: The physical system is the ground state of the one-dimensional transverse-field Ising model (TFIM) described by the Hamiltonian

$$H = -J \sum_{\langle ij \rangle} \sigma_i^z \sigma_j^z - h \sum_i \sigma_i^x \,, \tag{7.1}$$

where $\sigma^{x,y,z}$ are Pauli operators, defined over N sites (or qubits), and $\langle ij \rangle$ denotes nearest-neighbor pairs on a one-dimensional lattice with open boundary conditions. The RBMs are implemented with QuCumber and trained on projective measurement data generated with DMRG simulations (Ferris and Vidal, 2012; Fishman et al., 2020) for various system sizes and transverse field strengths h/J. We train the RBM in two distinct regimes: In the disordered *paramagnetic (PM) phase* with $h/J > 1$, and at the *quantum critical point (QCP)* where $h/J = 1$.

In general, for a fixed number of visible units N, the expressivity of the RBM is modified by varying the number of hidden units N_h, which is usually treated as a hyper-parameter and fixed at the beginning of training. Based on previous study (Sehayek et al., 2019), we use the ratio $N_h = N/2$ in current experiments.

We use iterative magnitude-based pruning and fine-tuning, meaning that weights are removed in steps, and the model is trained for several more epochs after each pruning step. The fine-tuning allows the remaining weights to adjust their values in order to compensate for the pruned weights. Starting from the trained model, we remove 40% of all weights in the first pruning step, and 5% of the remaining weights in all subsequent steps. We chose these parameters after testing a few schedules and finding that these moderate rates allow to obtain functioning RBMs with the least number of weights. Importantly, note that we prune only the weights, not the biases. We refer to the trained RBM before pruning as "dense" (as opposed to "sparse" after pruning).

The objective of the training procedure is the minimization of the *KL divergence* between the target distribution q estimated from the dataset, and the RBM distribution p_λ:

$$D_{\mathrm{KL}}(q \| p_\lambda) = \sum_{\boldsymbol{\sigma}} q(\boldsymbol{\sigma}) \ln \frac{q(\boldsymbol{\sigma})}{p_\lambda(\boldsymbol{\sigma})}. \tag{7.2}$$

In general, we cannot expect to reach a KL divergence of zero, yet because the KL is not calibrated and unbounded from above, any target threshold we set for the KL would be arbitrary. Therefore, in addition to the KL divergence, we introduce error measures for various physical observables and train the RBM for a fixed number of epochs ensuring that all measured errors have fully equilibrated. Measurements are made every 100 training epochs.

The KL divergence measures the discrepancy between the RBM-learned and the target distribution. Similarly, the *fidelity* measures the agreement between the reconstructed state and the exact wavefunction. For pure states,

$$F\left(\psi_\lambda, \Psi\right) = \left|\langle\psi_\lambda|\Psi\rangle\right|^2 . \tag{7.3}$$

Two characteristic observables for spin systems are the *energy* $E = \langle H\rangle$ and the *magnetization* along the z-axis, defined as

$$m = M/N = \sum_i^N \sigma_i^z, \tag{7.4}$$

which serves as the order parameter. Since the training data are simulated configurations of a finite-size system, the $\mathbb{Z}_2$ (spin-inversion) symmetry of the Hamiltonian is not broken in the ordered phase. Consequently, positive and negative contributions to $\langle m\rangle$ would cancel out. We therefore compute both $\langle m\rangle$ and the $\mathbb{Z}_2$-invariant quantity $\langle|m|\rangle$. As described in chapter 6, we measure the *relative observable error* (ROE) for the energy and the absolute magnetization, defined as

$$\mathrm{ROE} = \max\left|\frac{O_{\mathrm{DMRG}} - \bar{O}_{\mathrm{RBM}}}{O_{\mathrm{DMRG}}}\right|, \tag{7.5}$$

where O_{DMRG} is the DMRG expectation value, and $\bar{O}_{\mathrm{RBM}}$ is the RBM estimator with a statistical error correction calculated as $\bar{O}_{\mathrm{RBM}} = \langle O\rangle_{\mathrm{RBM}} \pm c\omega/\sqrt{n}$, where n is the number of samples, ω is the standard deviation, and $c = 2.576$ is a constant corresponding to 99% confidence interval. We shall use the short forms *eROE* and *mROE* for energy and magnetization ROE respectively. For more details on ROE, see equation 6.2 in chapter 6. The *two-point correlation function* is another important observable, in particular the functional form of its decay with increasing spin distance is a distinctive feature of a physical phase: We expect algebraic decay at the QCP, and exponential in the PM phase. We compute

$$C(d_{i,j}) = \langle\sigma_i^z\sigma_j^z\rangle - \langle\sigma_i^z\rangle\langle\sigma_j^z\rangle, \tag{7.6}$$

where d_{ij} is the distance between the sites i and j. To summarize the deviation between the two-point correlation function computed on spin configurations sampled from the RBM and the one computed on training data in a single scalar quantity, we use the *mean squared error*

$$C_{\mathrm{MSE}} = \sum_{j=\bar{i}}^{N} C(d_{\bar{i},j}) \,, \tag{7.7}$$

where $\bar{i} = N/2$ is the lattice midpoint, and the sum extends over all spin distances from 0 to $N/2$, as we are working with open boundary conditions. All expectation values are computed on 1e5 configurations drawn from a trained RBM (the same number of configurations as in the training set).

7.4 Results

We present the main results for a system of size $N = 18$, because this is the maximal size for which the computation of KL divergence and state fidelity is feasible. For larger systems, we compute only the eROE, mROE and the 2-point correlation functions, and find no qualitative difference to the smaller system.

Figure 7.2 demonstrates how various quantities that measure model quality evolve in the course of training and pruning. While convergence to the equilibrated value occurs within the first 500 training epochs for all observables, their degradation in response to pruning varies and is strongly dependent on the physical regime: RBM trained to model TFIM at the QCP takes damage already after a few pruning iterations, when the number of weights is reduced to about 50%. In contrast, the model for the PM regime does not show increase in any of the error measures until about 75% of weights have been removed.

The order parameter is the most pruning-sensitive observable. At the QCP, mROE increases from the first pruning iteration on, indicating that pruning has immediate effect on the order and, by extension, on the correlations between the spins. We take a closer look at the magnetization in Figure 7.3: The training set has $\langle m \rangle \approx 0$ and $\langle |m| \rangle \approx 0.5$, indicating the presence of (some) order and $\mathbb{Z}_2$ symmetry in the sample, and the histogram of m for the individual configurations shows a double-peak structure with maxima at $m \pm 0.75$, but also a significant proportion at $m = 0$. The sample set drawn from the trained dense RBM resembles the properties of the training set closely. However, in the course of pruning, the double-peak structure morphs into an increasingly pronounced single peak at $m = 0$, with $\mathbb{Z}_2$ symmetry preserved.

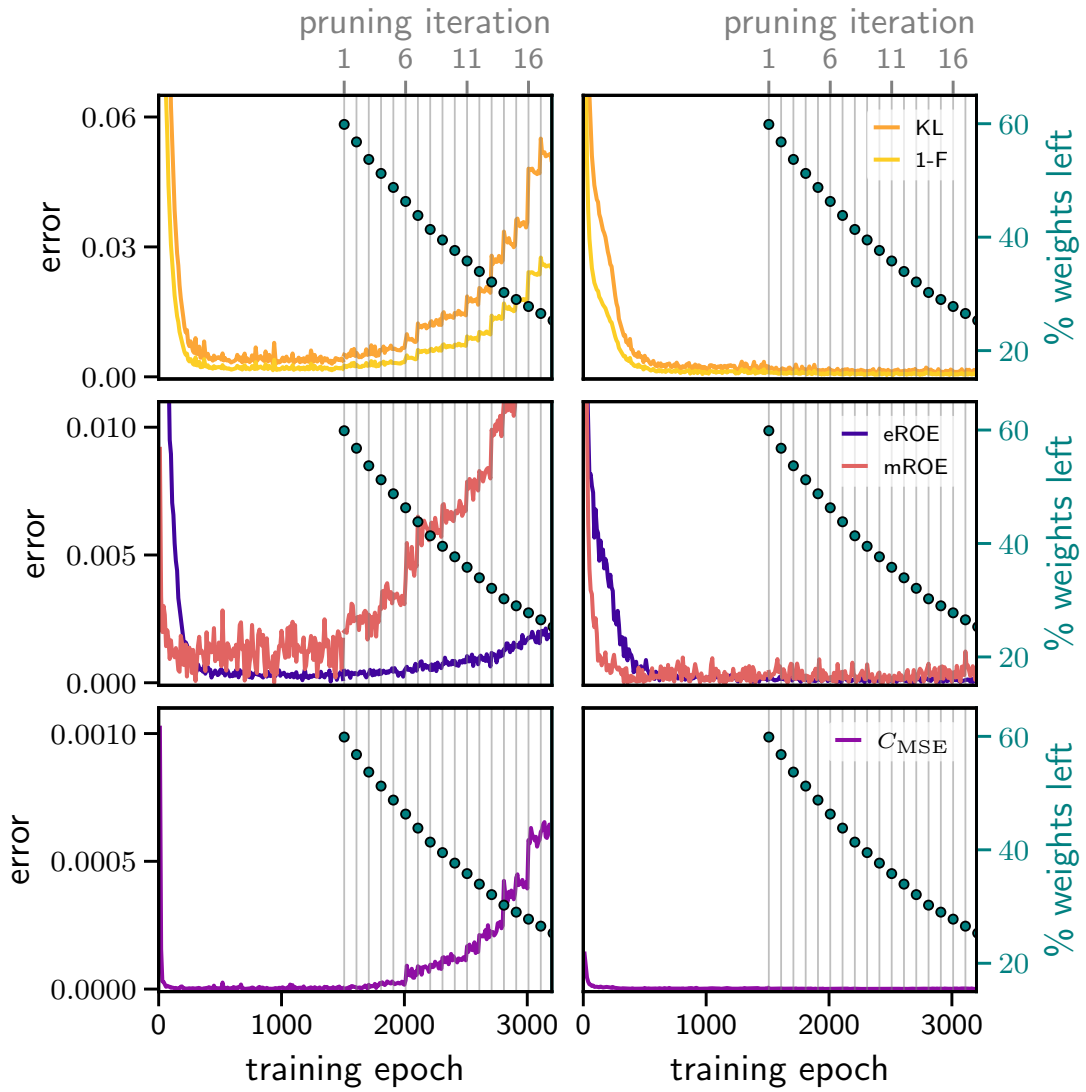


Figure 7.2: The behavior of various model quality measures during training and iterative pruning of an RBM trained at the quantum critical point at $h/J = 1.0$ (left), or in the paramagnetic regime at $h/J = 2.0$ (right). Vertical lines indicate pruning steps, labeled on the top x-axis. Secondary y-axis on the right shows the percentage of nonzero weights remaining in the RBM at every pruning iteration. *Top:* KL divergence of the learned RBM distribution and the quantum state fidelity (shown as 1-F). *Middle:* The relative observable error in energy and absolute magnetization (mROE scaled by factor 0.1). *Bottom:* The mean squared error of the spin-spin correlation function.

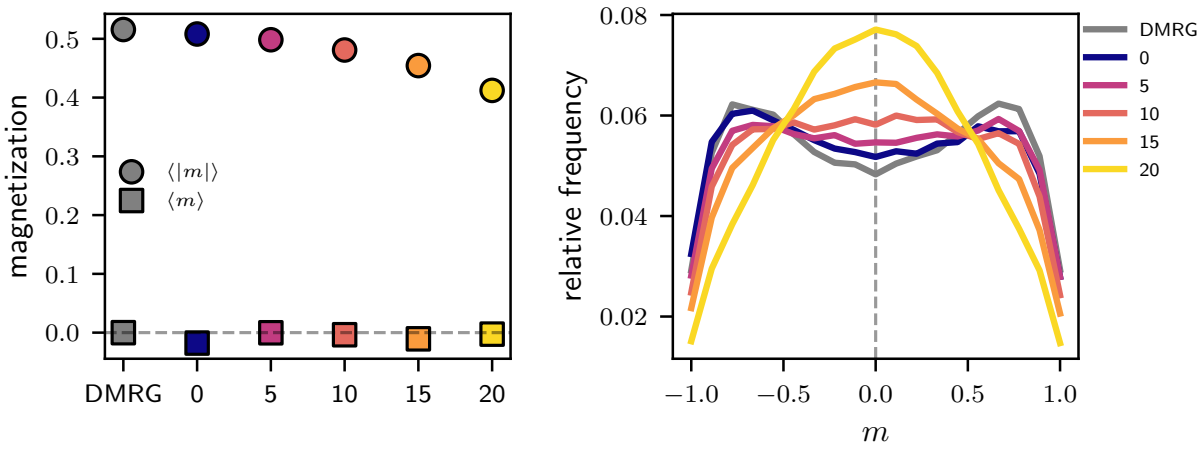


Figure 7.3: Magnetization expectation values $\langle m \rangle$ and $\langle |m| \rangle$ (left) and histogram of m values for each configuration in the sample (right) at the QCP. The numbers on the x-axis in the left plot and in the legend on the right plot indicate pruning iterations, where 0 stands for the trained dense RBM.

Similarly, in Figure 7.4 we show that pruning progressively destroys correlations between spins, changing the functional form of the two-point correlation function (CF) from algebraic to exponential decay. These results align with our intuition that the physical system is most complex at the QCP, where the correlation length spans the entire system and the configurations have some order. We therefore expect that more parameters are required to model a system at the QCP. In contrast, no long-range correlations exist in the disordered PM phase, thus the model can be simpler, and many weights in the RBM are redundant.

Note the effect of system size: In larger models, a larger percentage of weights has to be pruned in order to induce the same degree of damage to the CF. The reason for why larger RBMs seem more robust can be understood with the following combinatorial argument: An RBM with N visible and $N/2$ hidden units is a bipartite graph with $3N/2$ nodes in total. The minimal number of edges required to keep such graph connected is one less than nodes. For system sizes of 18, 50 or 100 spins (as shown in Figure 7.4) this number is 26, 74 or 149, corresponding to 16%, 6% or 3% of the total number of weights respectively. This means that at a given percentage of weights remaining, the graph of the smaller RBM is more likely to be disconnected than the graph of the larger RBM. In Figure 7.4, the numbers in the legend indicate pruning iterations, and each corresponds to a percentage of weights remaining in the RBM. Thus, for larger RBM, more pruning

iterations are required in order to achieve the same disconnectedness in the graph.

Finally, we examine the correlation between the KL divergence and the physical observables, in particular the error measures eROE, mROE and the mean squared error of the 2-point correlation function C_{MSE}. Figure 7.5 shows that the correlation with KL is different for each of these measures, and that it varies strongly depending on the stage of training or pruning. Specifically, KL is more sensitive to pruning than eROE, while mROE and C_{MSE} are much more sensitive than KL. Each of these accuracy measures captures a different relevant property of the system and thus a thorough evaluation of model quality has to include all of them.

7.5 Discussion

We have observed that the effect of pruning varies drastically depending on the interaction regime in the physical system: For RBMs trained to represent the quantum system in the paramagnetic state up to 50% of weights can be pruned without a significant loss of accuracy in the physical properties of the reconstructed state. In contrast, for RBMs trained on data at the quantum critical point, even a relatively small amount of pruning can have adverse effects on the model accuracy. This result is intuitive, as we know that the state of the physical system is most complex at the QCP, while the disordered PM phase is characterized by absence of spin-spin correlations. Therefore, we expect that the probability distribution that corresponds to the PM phase can be encoded in a simpler function, and thus many weights in the dense RBM are redundant. This result may have consequences beyond quantum critical systems, as much of the data from the natural world used to train neural networks in industry displays signatures of power-law decay.

Furthermore, our experiments have demonstrated that pruning has disparate effects on various model quality measures. Specifically, we have found that among all physical observables energy is least sensitive to pruning. More precisely, a pruned RBM can generate samples that have a reasonably accurate energy, while spin order and correlations show strong deviations from the ground truth. Thus, in our learning task, energy can not serve as a measure for model quality when pruning is applied. In general, our result demonstrates that it is important to monitor all relevant model quality measures instead of relying on a single observable. This is rather easy to accomplish when modeling a physical system, as the relevant physical observables are well-defined, but much harder for non-physical learning tasks, such as classification of natural images, where apart from the empirical test error no other measure of model quality is readily available. The danger is that pruning, while keeping the standard test accuracy intact, might be introducing unnoticed instabilities into

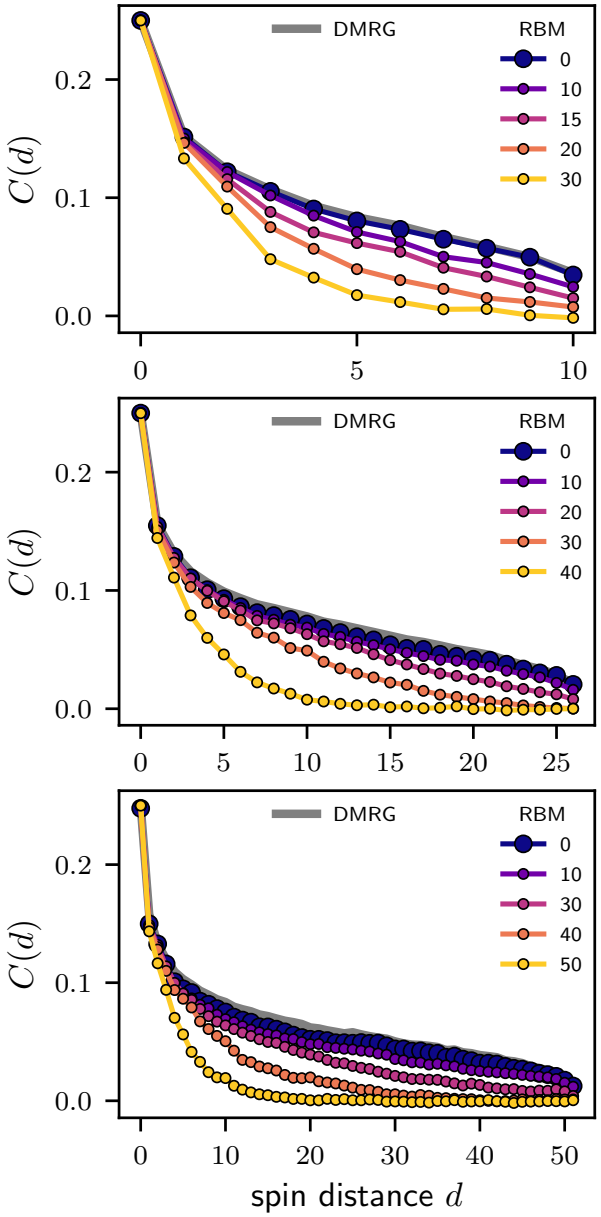


Figure 7.4: Spin-spin correlation function vs spin distance measured from the middle of the spin chain, for different system sizes (*top:* 18; *middle:* 50; *bottom:* 100 spins). Numbers in the legend indicate pruning iteration, where 0 is the trained dense RBM.

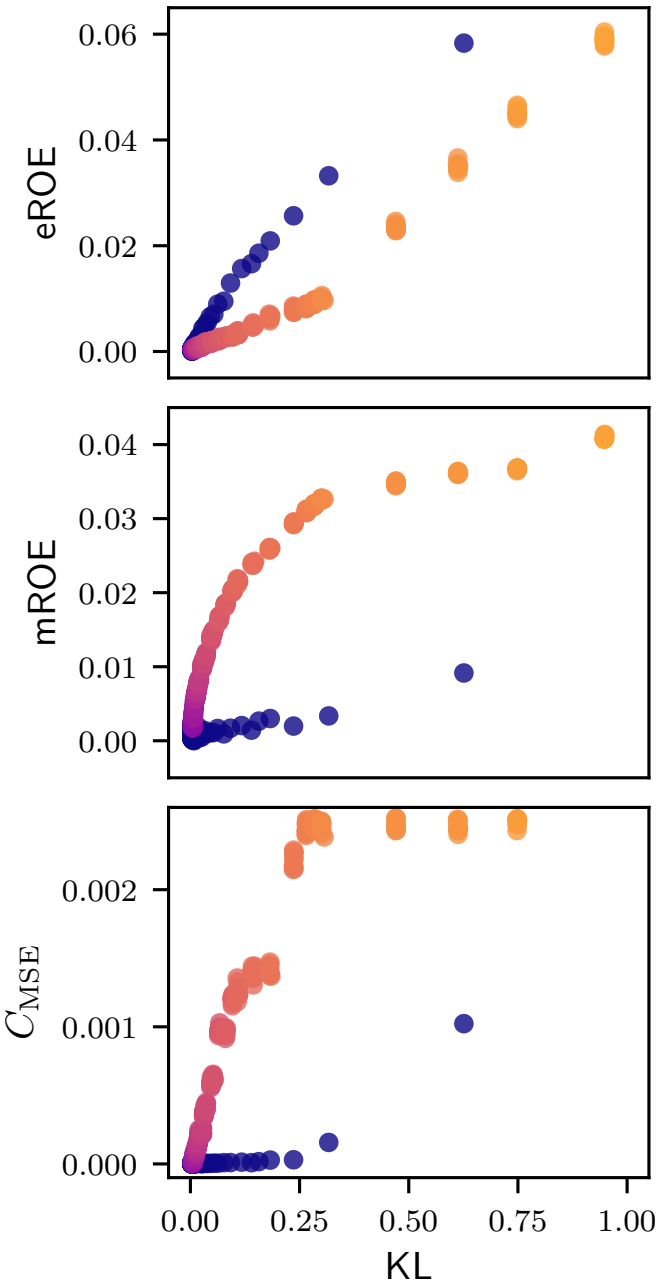


Figure 7.5: Correlation between physical error measures and the KL divergence during training and pruning. Training phase is indicated by blue markers, pruning phase by a color gradient from purple to yellow.

the model that might be exploited by adversaries, or lead to biased outputs. Indeed, in ML context it is known that pruning significantly reduces robustness to image corruptions and adversarial attacks[1] (Xiao et al., 2018; Guo et al., 2019). Furthermore, it was found that model performance is disproportionally impacted for classes of images that are generally more challenging to learn, which presents a concern for the fairness of AI algorithms.

We hope that our study will spark more research along the same lines, exploring the properties of NN-based ML models with the goal of making them a more understandable and principled tool for both scientific and industrial applications.

[1] An "adversarial attack" is an input image that was deliberately modified by introducing a small, typically unnoticeable variation, which causes a well-trained NN-based image classifier to make unreasonably wrong predictions.

Chapter 8

Discussion and Conclusion

In this thesis, I have presented several works that contribute towards building a better understanding of DL and explore the intersection between DL and quantum many-body physics. In one of these works, we have studied the role of layer width and parameter count, which is central for modern DL architectures, and exploited the relationship between NNs and Gaussian Processes to conduct a theoretical analysis in the limit of infinite width. In another line of work, we focused on generative modeling with a shallow NN architecture – the RBM – and studied its learning performance for physical systems. Based on the task of quantum state reconstruction, we have systematically analyzed the scaling of RBM learning and quantitatively characterized the relationships between elements of the learning model and the physical system. This investigation also included an analysis of pruning – a highly relevant technique for model compression in modern DL – which revealed a clear dependence of the attainable degree of pruning on the complexity of the data, and a strong disparity between various model quality measures in their sensitivity towards pruning.

My choice of research questions has been driven by the desire to understand DL on a more fundamental level, with the ultimate goal of making it a more principled and reliable tool, in particular for applications in science. As I described in chapters 1 and 2, DL is an immensely powerful novel technique that has proven successful in practice, with the caveat that the associated approaches and design principles for DL systems are not theoretically grounded. The lack of a theoretical foundation, the multitude of unexplained phenomena and the often counterintuitive combination of properties exhibited by DL systems demand caution in deployment of DL in critical settings.

The experience I gained in working on these projects has illuminated several aspects of DL that are of paramount importance for future progress. Perhaps the most fundamental of

them is the **role of data**. Despite the fact that DL is an inherently data-driven technique, the role of data remains understudied in current DL research. In the introduction chapter 1 I have mentioned the challenges associated with the empirical datasets used in DL: They are typically automatically collected from the Internet and their enormous size makes curation by human reviewers prohibitively expensive. As a result, datasets can harbor undetected abnormalities, such as mislabeled or ambiguous instances, or instances that by human judgment are not attributable to any of the given labels.

Apart from the issues related to dataset *characterization*, there are several more profound difficulties that could not be resolved even with unlimited human supervision. Specifically, consider the field of computer vision: DL systems excel at image recognition tasks, yet at the same time their performance can be dramatically affected by slight modifications of the input image – a feature exploited by *"adversarial attacks"*, as I mentioned in the previous chapter 7.5. Another major concern is the *"fairness"* of DL systems, as it has been noted that their performance in classification tasks can be non-uniform among image classes, i.e., DL systems can be *biased*. The central unresolved question in this context is whether the root cause of such bias lies in the training data alone or can be induced by DL algorithms as well. Considering that DL systems for image recognition are deployed in a variety of fields including autonomous driving, medical diagnostics and law enforcement, this brittleness and possible bias constitute serious potential threats.

Moreover, accounting for the dataset in the characterization of a learning system is key to *DL theory*. The discussion of the current state of DL theory and the associated challenges in section 2.1 mentions one of the critical failures of classical statistical learning theory when applied to DL: Its theory of generalization fully disregards the properties of the dataset. Meanwhile, empirical results in modern DL, but also our pruning study of "shallow" learning with RBMs in chapter 7, clearly demonstrate the necessity of including the dataset into consideration when designing learning systems or analyzing their behavior.

The biggest obstacle in understanding the role of data in DL is the lack of suitable well-defined metrics for the characterization of typical empirical datasets. In theory, we think of any kind of data as being drawn from some *generating distribution*. The challenge in practice is to define a distribution underlying images of objects – or subjects, such as people or cats. Is there a well-defined "cat distribution", and how does it differ from a "dog distribution"? The current state of research does not offer a definitive practical answer. It is evident that many important aspects of DL systems crucially depend on data. In order to make them accessible to investigations, we need to dedicate research efforts to data itself and identify both the relevant properties and appropriate analysis methods.

One viable way to address these data-related challenges is to study DL in application

to physics problems. The advantage of learning tasks related to physical systems is the availability of datasets and the associated metrics that characterize them. In particular the datasets obtained from quantum many-body systems are typically high-dimensional and complex, yet controlled. Specifically, I am referring to datasets that are obtained by sampling from simulated quantum systems, such as the transverse-field Ising model that was used in several research projects presented in this thesis. In such cases, we derive useful information about the data-generating distribution from the Hamiltonian that describes the physical system. Furthermore, our knowledge of the physical system and its behavior provides additional ways to test the validity of predictions obtained with a DL tool. For instance, in the RBM pruning project presented in chapter 7, we use a number of physical observables as metrics for model quality and find that they can be more sensitive indicators than the standard measure for model accuracy.

Another topic that deserves attention is the general approach to DL theory and how theoretical physics can be useful in this endeavor. As discussed in chapter 2, a rigorous theoretical treatment of DL systems is limited by their inherent complexity. Most impactful have been targeted experimental studies that reveal specific phenomena and analyze their causal relationships with certain elements of a DL system. However, such empirical investigations uncover *inter-dependencies*, but typically do not explain the *underlying cause*. Therefore, I suggest that the next step should be to supplement such analyzes with a theoretical investigation of a suitable *toy model*. This is one of the fundamental research principles in theoretical physics, and while finding a toy model that captures the effect of interest is a nontrivial task in itself, it is arguably indispensable for developing a precise characterization of any phenomenon. The analysis of *limiting cases*, as discussed in section 2.1.1, is another standard approach used in theoretical physics that is tremendously useful when attempting to understand the behavior of a system. For instance, the approach applied in the study presented in chapter 3 was developed with these guiding principles in mind: We combined systematic experiments on standard DL systems, which allowed to isolate a specific phenomenon in practice, with a theoretical analysis of a possible underlying cause in a suitable toy-NN model.

The shortcoming of the majority of all theoretical DL studies is that the proposed treatment applies to either strongly simplified or very specific settings, and therefore remains detached from practical DL. The development of more realistic toy models of both deep NNs and training data is therefore crucial for progress in DL theory. Similarly, the takeaway from works that focus on *analogies* between physics or other fields and DL is that it is imperative to judiciously evaluate the validity and value of imported ideas in DL context, bearing in mind that the primary purpose of DL is applications.

A final point regarding progress in DL theory: The current diversity of approaches

generates a multitude of disparate results. A necessary next step therefore is the consolidation of the accumulating knowledge that would aid the crystallization of a theory. In physics, the interplay of experiments and theory has been the driving force for conceptual advances. Thus, I believe that a thoughtful combination of theoretical analysis and controlled experiments targeted to a particular phenomenon in DL is the way forward.

Overall, it is fair to say that the current state of DL theory research – physics-inspired or otherwise – is at its exploration stage. As all beginnings, this is an exciting state to be in, and it offers plenty of opportunities for those willing to contribute.

Letters of Copyright Permission

American Physical Society
Reuse and Permissions License

19-Apr-2021

This license agreement between the American Physical Society ("APS") and Anna Golubeva ("You") consists of your license details and the terms and conditions provided by the American Physical Society and SciPris.

Licensed Content Information

License Number:	**RNP/21/APR/038971**
License date:	19-Apr-2021
DOI:	10.1103/PhysRevB.100.195125
Title:	Learnability scaling of quantum states: Restricted Boltzmann machines
Author:	Dan Sehayek et al.
Publication:	Physical Review B
Publisher:	American Physical Society
Cost:	USD $ 0.00

Request Details

Does your reuse require significant modifications: No

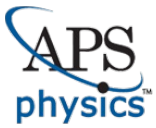
Specify intended distribution locations:	Worldwide
Reuse Category:	Reuse in a thesis/dissertation
Requestor Type:	Author of requested content
Items for Reuse:	Whole Article
Format for Reuse:	Electronic

Information about New Publication:

University/Publisher:	University of Waterloo
Title of dissertation/thesis:	no title yet
Author(s):	Anna Golubeva
Expected completion date:	Jul. 2021

License Requestor Information

Name:	Anna Golubeva
Affiliation:	Individual
Email Id:	agolubeva@perimeterinstitute.ca
Country:	Canada



American Physical Society
Reuse and Permissions License

TERMS AND CONDITIONS

The American Physical Society (APS) is pleased to grant the Requestor of this license a non-exclusive, non-transferable permission, limited to Electronic format, provided all criteria outlined below are followed.

1. You must also obtain permission from at least one of the lead authors for each separate work, if you haven't done so already. The author's name and affiliation can be found on the first page of the published Article.
2. For electronic format permissions, Requestor agrees to provide a hyperlink from the reprinted APS material using the source material's DOI on the web page where the work appears. The hyperlink should use the standard DOI resolution URL, http://dx.doi.org/{DOI}. The hyperlink may be embedded in the copyright credit line.
3. For print format permissions, Requestor agrees to print the required copyright credit line on the first page where the material appears: "Reprinted (abstract/excerpt/figure) with permission from [(FULL REFERENCE CITATION) as follows: Author's Names, APS Journal Title, Volume Number, Page Number and Year of Publication.] Copyright (YEAR) by the American Physical Society."
4. Permission granted in this license is for a one-time use and does not include permission for any future editions, updates, databases, formats or other matters. Permission must be sought for any additional use.
5. Use of the material does not and must not imply any endorsement by APS.
6. APS does not imply, purport or intend to grant permission to reuse materials to which it does not hold copyright. It is the requestor's sole responsibility to ensure the licensed material is original to APS and does not contain the copyright of another entity, and that the copyright notice of the figure, photograph, cover or table does not indicate it was reprinted by APS with permission from another source.
7. The permission granted herein is personal to the Requestor for the use specified and is not transferable or assignable without express written permission of APS. This license may not be amended except in writing by APS.
8. You may not alter, edit or modify the material in any manner.
9. You may translate the materials only when translation rights have been granted.
10. APS is not responsible for any errors or omissions due to translation.
11. You may not use the material for promotional, sales, advertising or marketing purposes.
12. The foregoing license shall not take effect unless and until APS or its agent, Aptara, receives payment in full in accordance with Aptara Billing and Payment Terms and Conditions, which are incorporated herein by reference.
13. Should the terms of this license be violated at any time, APS or Aptara may revoke the license with no refund to you and seek relief to the fullest extent of the laws of the USA. Official written notice will be made using the contact information provided with the permission request. Failure to receive such notice will not nullify revocation of the permission.
14. APS reserves all rights not specifically granted herein.
15. This document, including the Aptara Billing and Payment Terms and Conditions, shall be the entire agreement between the parties relating to the subject matter hereof.

References

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks, 2013.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(76):2493–2537, 2011. URL http://jmlr.org/papers/v12/collobert11a.html.

Ruhi Sarikaya, Geoffrey E. Hinton, and Anoop Deoras. Application of deep belief networks for natural language understanding. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 22(4):778–784, April 2014. ISSN 2329-9290. doi: 10.1109/TASLP.2014.2303296. URL https://doi.org/10.1109/TASLP.2014.2303296.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy

physics with deep learning. *Nature Communications*, 5(1), Jul 2014. ISSN 2041-1723. doi: 10.1038/ncomms5308. URL http://dx.doi.org/10.1038/ncomms5308.

Louis-François Arsenault, O. Anatole von Lilienfeld, and Andrew J. Millis. Machine learning for many-body physics: efficient solution of dynamical mean-field theory, 2015.

Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, Feb 2017. ISSN 1745-2481. doi: 10.1038/nphys4035. URL http://dx.doi.org/10.1038/nphys4035.

Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4), Dec 2019. ISSN 1539-0756. doi: 10.1103/revmodphys.91.045002. URL http://dx.doi.org/10.1103/RevModPhys.91.045002.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

François Chollet et al. Keras. https://keras.io, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

A. Engel and C. Van den Broeck. *Statistical Mechanics of Learning*. Cambridge University Press, 2001. ISBN 9780521773072. URL https://books.google.ca/books?id=0RZQtAEACAAJ.

Daniel Amit, Hanoch Gutfreund, and Haim Sompolinsky. Spin-glass models of neural networks. 32, 09 1985. doi: 10.1103/PhysRevA.32.1007.

Timothy L. H. Watkin, Albrecht Rau, and Michael Biehl. The statistical mechanics of learning a rule. *Rev. Mod. Phys.*, 65:499–556, Apr 1993. doi: 10.1103/RevModPhys.65.499. URL https://link.aps.org/doi/10.1103/RevModPhys.65.499.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks, 2015.

Lenka Zdeborová and Florent Krzakala. Statistical physics of inference: thresholds and algorithms. *Advances in Physics*, 65(5):453–552, Aug 2016. ISSN 1460-6976. doi: 10.1080/00018732.2016.1211393. URL http://dx.doi.org/10.1080/00018732.2016.1211393.

Cédric Bény. Deep learning and the renormalization group. *arXiv:1301.3124 [quant-ph]*, January 2013. URL http://arxiv.org/abs/1301.3124. arXiv: 1301.3124.

Pankaj Mehta and David J. Schwab. An exact mapping between the Variational Renormalization Group and Deep Learning. October 2014. URL http://arxiv.org/abs/1410.3831.

Satoshi Iso, Shotaro Shiba, and Sumito Yokoo. Scale-invariant feature extraction of neural network and renormalization group flow. *Phys. Rev. E*, 97:053304, May 2018. doi: 10.1103/PhysRevE.97.053304. URL https://link.aps.org/doi/10.1103/PhysRevE.97.053304.

Maciej Koch-Janusz and Zohar Ringel. Mutual information, neural networks and the renormalization group. *Nature Physics*, 14:578, June 2018. ISSN 1745-2481. URL https://www.nature.com/articles/s41567-018-0081-4.

Shuo-Hui Li and Lei Wang. Neural Network Renormalization Group. February 2018. URL https://arxiv.org/abs/1802.02840v3.

Shotaro Shiba Funai and Dimitrios Giataganas. Thermodynamics and feature extraction by machine learning. *Physical Review Research*, 2(3), Sep 2020. ISSN 2643-1564. doi: 10.1103/physrevresearch.2.033415. URL http://dx.doi.org/10.1103/PhysRevResearch.2.033415.

Ellen De Mello Koch, Robert De Mello Koch, and Ling Cheng. Is deep learning a renormalization group flow? *IEEE Access*, 8:106487–106505, 2020. ISSN 2169-3536. doi: 10.1109/access.2020.3000901. URL http://dx.doi.org/10.1109/ACCESS.2020.3000901.

Ethan Dyer and Guy Gur-Ari. Asymptotics of wide networks from feynman diagrams, 2019.

Omry Cohen, Or Malka, and Zohar Ringel. Learning curves for overparametrized deep neural networks: A field theory perspective. *Physical Review Research*, 3(2), Apr 2021. ISSN 2643-1564. doi: 10.1103/physrevresearch.3.023034. URL http://dx.doi.org/10.1103/PhysRevResearch.3.023034.

Joseph M. Antognini. Finite size corrections for neural network gaussian processes, 2019.

Sho Yaida. Non-gaussian processes and neural networks at finite widths, 2020.

James Halverson, Anindita Maiti, and Keegan Stoner. Neural networks and quantum field theory. *Machine Learning: Science and Technology*, Mar 2021. ISSN 2632-2153. doi: 10.1088/2632-2153/abeca3. URL http://dx.doi.org/10.1088/2632-2153/abeca3.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes, 2018a.

Alexander G. de G. Matthews, Mark Rowland, Jiri Hron, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks, 2018.

Daniel Kunin, Javier Sagastuy-Brena, Surya Ganguli, Daniel L. K. Yamins, and Hidenori Tanaka. Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics, 2021.

Greg Yang and Samuel S. Schoenholz. Mean field residual networks: On the edge of chaos, 2017.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of batch normalization, 2019.

Dar Gilboa, Bo Chang, Minmin Chen, Greg Yang, Samuel S. Schoenholz, Ed H. Chi, and Jeffrey Pennington. Dynamical isometry and a mean field theory of lstms and grus, 2019.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

Yasaman Bahri, Jonathan Kadmon, Jeffrey Pennington, Sam S. Schoenholz, Jascha Sohl-Dickstein, and Surya Ganguli. Statistical mechanics of deep learning. *Annual Review of Condensed Matter Physics*, 11(1):501–528, 2020. doi: 10.1146/annurev-conmatphys-031119-050745. URL https://doi.org/10.1146/annurev-conmatphys-031119-050745.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2017a.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off, 2019.

Radford M Neal. Priors for infinite networks. 1994.

Christopher K. I. Williams. Computation with Infinite Neural Networks. *Neural Computation*, 10(5):1203–1216, 07 1998. ISSN 0899-7667. doi: 10.1162/089976698300017412.

Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes, 2019.

Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes, 2020.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. Gpflow: A gaussian process library using tensorflow. *Journal of Machine Learning Research*, 18 (40):1–6, 2017. URL http://jmlr.org/papers/v18/16-537.html.

Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences, 2018.

Julius Berner, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. The modern mathematics of deep learning, 2021.

Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995. ISBN 0-387-94559-8.

Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. *Introduction to Statistical Learning Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-28650-9. doi: 10.1007/978-3-540-28650-9_8. URL https://doi.org/10.1007/978-3-540-28650-9_8.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Stefan Leijnen and Fjodor Veen. The neural network zoo. *Proceedings*, 47:9, 05 2020. doi: 10.3390/proceedings47010009.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, USA, 2004. ISBN 0521813972.

Fanghui Liu, Xiaolin Huang, Yudong Chen, and Johan A. K. Suykens. Random features for kernel approximation: A survey on algorithms, theory, and beyond, 2021.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18:1527–1554, 2006. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.2006.18.7.1527#.VzSfdWamvEY.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

CE Rasmussen and CKI Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

Anna Golubeva, Guy Gur-Ari, and Behnam Neyshabur. Are wider nets better given the same number of parameters? In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=_zx8Oka09eF.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth, 2021.

Vaggos Chatziafratis, Sai Ganesh Nagarajan, Ioannis Panageas, and Xiao Wang. Depth-width trade-offs for relu networks via sharkovsky's theorem, 2019.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR (Workshop)*, 2015. URL http://arxiv.org/abs/1412.6614.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BygfghAcYX.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8141–8150, 2019.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378, 2016.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65(2):742–769, 2018.

Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. *arXiv preprint arXiv:2002.09277*, 2020.

Jaehoon Lee, Samuel S. Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite Versus Infinite Neural Networks: an Empirical Study. *arXiv e-prints*, art. arXiv:2007.15801, July 2020.

Etai Littwin, Ben Myara, Sima Sabah, Joshua Susskind, Shuangfei Zhai, and Oren Golan. Collegial ensembles, 2020.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv e-prints*, art. arXiv:1611.05431, November 2016.

Jongsoo Park, Sheng R. Li, Wei Wen, Hai Li, Yiran Chen, and Pradeep Dubey. Holistic sparsecnn: Forging the trident of accuracy, speed, and size. *ArXiv*, abs/1608.01409, 2016.

Sharan Narang, Greg Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. *ArXiv*, abs/1704.05119, 2017.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert A. Legenstein. Deep rewiring: Training very sparse deep networks. *ArXiv*, abs/1711.05136, 2018.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL http://arxiv.org/abs/1803.03635.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. *ArXiv*, abs/1911.09723, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *ArXiv*, abs/1902.09574, 2019.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks, 2019.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: single-shot network pruning based on connection sensitivity. *CoRR*, abs/1810.02340, 2018b. URL http://arxiv.org/abs/1810.02340.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners, 2019.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow, 2020.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow, 2020.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark?, 2020.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Matthew J. S. Beach, Isaac De Vlugt, Anna Golubeva, Patrick Huembeli, Bohdan Kulchytskyy, Xiuzhe Luo, Roger G. Melko, Ejaaz Merali, and Giacomo Torlai. QuCumber: wavefunction reconstruction with neural networks. *SciPost Phys.*, 7:9, 2019a. doi: 10.21468/SciPostPhys.7.1.009. URL https://scipost.org/10.21468/SciPostPhys.7.1.009.

P. Smolensky. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed Processing*, chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pages 194–281. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL http://dl.acm.org/citation.cfm?id=104279.104290.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14:1771–1800, 2002.

Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Comput.*, 20(6):1631–1649, June 2008. ISSN 0899-7667. doi: 10.1162/neco.2008.04-07-510. URL http://dx.doi.org/10.1162/neco.2008.04-07-510.

Giacomo Torlai. *Augmenting Quantum Mechanics with Artificial Intelligence*. PhD thesis, University of Waterloo, 2018. URL http://hdl.handle.net/10012/14196.

Asja Fischer and Christian Igel. Bounding the bias of contrastive divergence learning. *Neural Computation*, 23(3):664–673, 2011. doi: 10.1162/NECO_a_00085.

Geoffrey E Hinton. A Practical Guide to Training Restricted Boltzmann Machines. In Grégoire Montavon, Geneviève B Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade: Second Edition*, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_32. URL https://doi.org/10.1007/978-3-642-35289-8{_}32http://link.springer.com/10.1007/978-3-642-35289-8{_}32.

T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1064–1071, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390290. URL http://doi.acm.org/10.1145/1390156.1390290.

Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549:242–246, September 2017. ISSN 1476-4687. URL https://www.nature.com/articles/nature23879.

Nikolaj Moll, Panagiotis Barkoutsos, Lev S. Bishop, Jerry M. Chow, Andrew Cross, Daniel J. Egger, Stefan Filipp, Andreas Fuhrer, Jay M. Gambetta, Marc Ganzhorn, Abhinav Kandala, Antonio Mezzacapo, Peter Müller, Walter Riess, Gian Salis, John Smolin, Ivano Tavernelli, and Kristan Temme. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Sci. Technol.*, 3:030503, 2018. ISSN 2058-9565. URL http://stacks.iop.org/2058-9565/3/i=3/a=030503.

Hannes Bernien, Sylvain Schwartz, Alexander Keesling, Harry Levine, Ahmed Omran, Hannes Pichler, Soonwon Choi, Alexander S. Zibrov, Manuel Endres, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. Probing many-body dynamics on a 51-atom quantum simulator. *Nature*, 551:579–584, November 2017. ISSN 1476-4687. URL https://www.nature.com/articles/nature24622.

J. Zhang, G. Pagano, P. W. Hess, A. Kyprianidis, P. Becker, H. Kaplan, A. V. Gorshkov, Z.-X. Gong, and C. Monroe. Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator. *Nature*, 551:601–604, November 2017b. ISSN 1476-4687. URL https://www.nature.com/articles/nature24654.

Giacomo Torlai and Roger G. Melko. Learning thermodynamics with Boltzmann machines. *Phys. Rev. B*, 94:165134, October 2016. URL https://link.aps.org/doi/10.1103/PhysRevB.94.165134.

Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14:447, 2018.

Giacomo Torlai and Roger G. Melko. Latent Space Purification via Neural Density Operators. *Phys. Rev. Lett.*, 120:240503, June 2018. URL https://link.aps.org/doi/10.1103/PhysRevLett.120.240503.

Juan Carrasquilla, Giacomo Torlai, Roger G. Melko, and Leandro Aolita. Reconstructing quantum states with generative models. *Nature Machine Intelligence*, 1(3): 155–161, 2019. doi: 10.1038/s42256-019-0028-1. URL https://doi.org/10.1038/s42256-019-0028-1.

D. T. Lennon, H. Moon, L. C. Camenzind, Liuqi Yu, D. M. Zumbühl, G. A. D. Briggs, M. A. Osborne, E. A. Laird, and N. Ares. Efficiently measuring a quantum device using machine learning. October 2018. URL http://arxiv.org/abs/1810.10042.

C. Kim, J. K. Rhee, W. Lee, and J. Ahn. Mixed Quantum State Dynamics Estimation with Artificial Neural Network. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 740–747, October 2018.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006. ISSN 0036-8075. URL http://science.sciencemag.org/content/313/5786/504.

Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL https://doi.org/10.1038/nature14539.

Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355:602–606, 2017a. ISSN 0036-8075. URL http://science.sciencemag.org/content/355/6325/602.

Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. Constructing exact representations of quantum many-body systems with deep neural networks. *Nature Communications*, 9:5322, December 2018a. ISSN 2041-1723. URL https://www.nature.com/articles/s41467-018-07520-3.

Xun Gao and Lu-Ming Duan. Efficient representation of quantum many-body states with deep neural networks. *Nature Communications*, 8:662, September 2017a. ISSN 2041-1723. URL https://www.nature.com/articles/s41467-017-00705-2.

Kenny Choo, Giuseppe Carleo, Nicolas Regnault, and Titus Neupert. Symmetries and Many-Body Excitations with Neural-Network Quantum States. *Phys. Rev. Lett.*, 121: 167204, October 2018. URL https://link.aps.org/doi/10.1103/PhysRevLett.121.167204.

Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. Neural-Network Quantum States, String-Bond States, and Chiral Topological States.

Phys. Rev. X, 8:011006, January 2018a. URL https://link.aps.org/doi/10.1103/PhysRevX.8.011006.

Patrick M. Lenggenhager, Zohar Ringel, Sebastian D. Huber, and Maciej Koch-Janusz. Optimal Renormalization Group Transformation from Information Theory. September 2018. URL http://arxiv.org/abs/1809.09632.

Jing Chen, Song Cheng, Haidong Xie, Lei Wang, and Tao Xiang. Equivalence of restricted boltzmann machines and tensor network states. *Physical Review B*, 97(8), Feb 2018. ISSN 2469-9969. doi: 10.1103/physrevb.97.085104. URL http://dx.doi.org/10.1103/PhysRevB.97.085104.

Yusuke Nomura, Andrew S. Darmawan, Youhei Yamaji, and Masatoshi Imada. Restricted boltzmann machine learning for solving strongly correlated quantum systems. *Phys. Rev. B*, 96:205152, Nov 2017. URL https://link.aps.org/doi/10.1103/PhysRevB.96.205152.

Steven Weinstein. Neural networks as "hidden" variable models for quantum systems. July 2018. URL http://arxiv.org/abs/1807.03910.

Yunqin Zheng, Huan He, Nicolas Regnault, and B. Andrei Bernevig. Restricted boltzmann machines and matrix product states of 1d translational invariant stabilizer codes. 2018.

Giacomo Torlai and Roger G. Melko. Machine-learning quantum states in the NISQ era. *Annual Review of Condensed Matter Physics*, 11(1):325–344, Mar 2020. ISSN 1947-5462. doi: 10.1146/annurev-conmatphys-031119-050651. URL http://dx.doi.org/10.1146/annurev-conmatphys-031119-050651.

Matthew B. Hastings, Iván González, Ann B. Kallin, and Roger G. Melko. Measuring renyi entanglement entropy in quantum monte carlo simulations. *Phys. Rev. Lett.*, 104:157201, Apr 2010. URL https://link.aps.org/doi/10.1103/PhysRevLett.104.157201.

Dan Sehayek, Anna Golubeva, Michael S. Albergo, Bohdan Kulchytskyy, Giacomo Torlai, and Roger G. Melko. Learnability scaling of quantum states: Restricted boltzmann machines. *Phys. Rev. B*, 100:195125, Nov 2019. doi: 10.1103/PhysRevB.100.195125. URL https://link.aps.org/doi/10.1103/PhysRevB.100.195125.

Song Cheng, Jing Chen, and Lei Wang. Information perspective to probabilistic modeling: Boltzmann machines versus born machines. *Entropy*, 20(8), 2018. ISSN 1099-4300. doi: 10.3390/e20080583. URL https://www.mdpi.com/1099-4300/20/8/583.

Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. Constructing exact representations of quantum many-body systems with deep neural networks. *Nature Communications*, 9(1):5322, 2018b. doi: 10.1038/s41467-018-07520-3. URL https://doi.org/10.1038/s41467-018-07520-3.

Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017b. ISSN 0036-8075. doi: 10.1126/science.aag2302. URL http://science.sciencemag.org/content/355/6325/602.

Giacomo Torlai, Brian Timar, Evert P.L. van Nieuwenburg, Harry Levine, Ahmed Omran, Alexander Keesling, Hannes Bernien, Markus Greiner, Vladan Vuletić, Mikhail D. Lukin, Roger G. Melko, and Manuel Endres. Integrating neural networks with a quantum simulator for state reconstruction. 2019.

Roger G. Melko, Giuseppe Carleo, Juan Carrasquilla, and J. Ignacio Cirac. Restricted boltzmann machines in quantum physics. *Nature Physics*, 2019. URL https://doi.org/10.1038/s41567-019-0545-1.

Stefanie Czischek, Jan M. Pawlowski, Thomas Gasenzer, and Martin Garttner. Sampling scheme for neuromorphic simulation of entangled quantum systems. 2019.

Matteo Paris and Jaroslav Rehacek, editors. *Quantum State Estimation*, volume 649 of *Lecture Notes in Physics*. Springer-Verlag Berlin Heidelberg, 2004.

Anders W. Sandvik. Stochastic series expansion method for quantum ising models with arbitrary interactions. *Phys. Rev. E*, 68:056701, Nov 2003. doi: 10.1103/PhysRevE.68.056701. URL https://link.aps.org/doi/10.1103/PhysRevE.68.056701.

E. M. Inack, G. Giudici, T. Parolini, G. Santoro, and S. Pilati. Understanding quantum tunneling using diffusion monte carlo simulations. *Phys. Rev. A*, 97:032307, Mar 2018. doi: 10.1103/PhysRevA.97.032307. URL https://link.aps.org/doi/10.1103/PhysRevA.97.032307.

G. Vidal. Entanglement renormalization. *Phys. Rev. Lett.*, 99:220405, Nov 2007. doi: 10.1103/PhysRevLett.99.220405. URL https://link.aps.org/doi/10.1103/PhysRevLett.99.220405.

Wen Wei Ho and Timothy H. Hsieh. Efficient variational simulation of non-trivial quantum states. *SciPost Phys.*, 6:29, 2019. doi: 10.21468/SciPostPhys.6.3.029. URL https://scipost.org/10.21468/SciPostPhys.6.3.029.

Wen Wei Ho, Cheryne Jonay, and Timothy H. Hsieh. Ultrafast variational simulation of nontrivial quantum states with long-range interactions. *Phys. Rev. A*, 99:052332, May 2019. doi: 10.1103/PhysRevA.99.052332. URL https://link.aps.org/doi/10.1103/PhysRevA.99.052332.

Matthew J. S. Beach, Roger G. Melko, Tarun Grover, and Timothy H. Hsieh. Making trotters sprint: A variational imaginary time ansatz for quantum many-body systems. 2019b.

Andrew J. Ferris and Guifre Vidal. Perfect sampling with unitary tensor networks. *Phys. Rev. B*, 85:165146, Apr 2012. doi: 10.1103/PhysRevB.85.165146. URL https://link.aps.org/doi/10.1103/PhysRevB.85.165146.

Matthew Fishman, Steven R. White, and E. Miles Stoudenmire. The ITensor software library for tensor network calculations, 2020.

Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117 – 158, 2014. ISSN 0003-4916. doi: https://doi.org/10.1016/j.aop.2014.06.013. URL http://www.sciencedirect.com/science/article/pii/S0003491614001596.

Román Orús. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1 (9):538–550, 2019. doi: 10.1038/s42254-019-0086-7. URL https://doi.org/10.1038/s42254-019-0086-7.

Scott Aaronson. The learnability of quantum states. *Proc. R. Soc. A*, 463, 2007.

David Lopez-Paz and Levent Sagun. Easing non-convex optimization with neural networks, 2018. URL https://openreview.net/forum?id=rJXIPK1PM.

Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. *CoRR*, abs/1410.1141, 2014. URL http://arxiv.org/abs/1410.1141.

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *CoRR*, abs/1802.06509, 2018. URL http://arxiv.org/abs/1802.06509.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. 2018.

Karthik A. Sankararaman, Soham De, Zheng Xu, W. Ronny Huang, and Tom Goldstein. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. 2019.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015a. URL http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf.

Decebal Constantin Mocanu, Elena Mocanu, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2):243–270, Sep 2016. ISSN 1573-0565. doi: 10.1007/s10994-016-5570-z. URL https://doi.org/10.1007/s10994-016-5570-z.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. 2019.

Xun Gao and Lu-Ming Duan. Efficient representation of quantum many-body states with deep neural networks. *Nature Communications*, 8(1):662, 2017b. doi: 10.1038/s41467-017-00705-2. URL https://doi.org/10.1038/s41467-017-00705-2.

Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. Neural-network quantum states, string-bond states, and chiral topological states. *Phys. Rev. X*, 8:011006, Jan 2018b. doi: 10.1103/PhysRevX.8.011006. URL https://link.aps.org/doi/10.1103/PhysRevX.8.011006.

Anton Mazurenko, Christie S. Chiu, Geoffrey Ji, Maxwell F. Parsons, Márton Kanász-Nagy, Richard Schmidt, Fabian Grusdt, Eugene Demler, Daniel Greif, and Markus Greiner. A cold-atom fermi–hubbard antiferromagnet. *Nature*, 545:462 EP –, 05 2017. URL https://doi.org/10.1038/nature22362.

B. P. Lanyon, C. Maier, M. Holzäpfel, T. Baumgratz, C. Hempel, P. Jurcevic, I. Dhand, A. S. Buyskikh, A. J. Daley, M. Cramer, M. B. Plenio, R. Blatt, and C. F. Roos. Efficient tomography of a quantum many-body system. *Nature Physics*, 13(12):1158–1162, sep 2017. ISSN 1745-2473. doi: 10.1038/nphys4244. URL http://www.nature.com/doifinder/10.1038/nphys4244.

Alexander Keesling, Ahmed Omran, Harry Levine, Hannes Bernien, Hannes Pichler, Soonwon Choi, Rhine Samajdar, Sylvain Schwartz, Pietro Silvi, Subir Sachdev, Peter Zoller,

Manuel Endres, Markus Greiner, Vladan Vuletic, and Mikhail D. Lukin. Probing quantum critical dynamics on a programmable Rydberg simulator. sep 2018. URL http://arxiv.org/abs/1809.05540.

R. Reed. Pruning algorithms – a survey. *IEEE Transactions on Neural Networks*, 4(5): 740–747, 1993. doi: 10.1109/72.248452.

Georg Thimm and Emile Fiesler. Evaluating pruning methods. In *National Chiao-Tung University*, page 2, 1995.

Nikko Ström. Sparse connection and pruning in large dynamic artificial neural networks. In *Fifth European Conference on Speech Communication and Technology*, 1997.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015b.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning, 2017.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient processing of deep neural networks: A tutorial and survey, 2017.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 2020.

Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples, 2018.

Yiwen Guo, Chao Zhang, Changshui Zhang, and Yurong Chen. Sparse dnns with improved adversarial robustness, 2019.

APPENDICES

Appendix A

Experimental details

In this section we provide experimental details and additional information about the figures in chapter 3.

A general description of the datasets used in this work can be found in the documentation of the pytorch package torchvision.

A.1 ImageNet data set

ImageNet data preprocessing: In order to decrease the size of the data set and be able to download it to cloud instances with limited storage, we resized all images in the data set by keeping their proportions fixed and setting their smallest dimension to 256. This procedure reduces the accuracy of ResNet models by about 1-2%.

Transformations for ImageNet: We apply standard transformations used for training on ImageNet:

- `RandomResizedCrop(size=224, scale=(0.2, 1.0))` on the training set.
- `Resize(256, transforms.CenterCrop(224))` on the test set.

A.2 ResNet-18 model

In all experiments, we use a standard PyTorch implementation of the ResNet-18 model. We set the number of weights in the model by changing the number of output channels in the first convolutional layer (referred to as *the* model width), while leaving the width ratios between the first convolutional layer and the subsequent four blocks of the ResNet-18 at their default values $1 : 2 : 4 : 8$. We do not include the weights in the BatchNorm layers into the total weight count, and we do not sparsify these layers.

A.3 Figures and experiments

Figures 3.1 and 3.6a: All models are trained using SGD with momentum=0.9, Cross-Entropy loss, and initial learning rate 0.1. The learning rate value and schedule were tuned for the smallest baseline model. We do not apply early stopping, and we report the best achieved test accuracy.

For **ImageNet**, we use weight decay 1e-4, cosine learning rate schedule, and train for 150 epochs. Because of the computational cost of ImageNet experiments, we did not repeat each run multiple times. We have checked for two data points that the variance in training and test accuracy for different random seeds is smaller than 0.1%.

For **other data sets**, we use weight decay 5e-4, train for 300 epochs, and the initial learning rate 0.1 is decayed at epochs 50, 120 and 200 with gamma=0.1. The reported results are averages over 3 runs with different random seeds.

In Figure 3.1, the baseline model has width 64 (1e7 weights) for ImageNet, and 18 (9e5 weights) for the other data sets.

In Figure 3.6a, we consider baseline models with base widths [8, 12, 18, 40, 64], corresponding to a total of [1.8e5, 4.0e5, 9.0e5, 4.4e6, 1.1e7] weights respectively.

Figure 3.5: All networks are MLPs with one hidden layer, a total of 3970 weights (base width is 5), and either (a) ReLU or (b) Linear activation function. The networks are parametrized according to the standard PyTorch implementation (weights and biases are randomly initialized from the uniform distribution). We train these models on MNIST for a fixed number of 300 epochs (ensuring convergence), with SGD optimizer, no momentum, Cross-Entropy loss, with a constant learning rate 0.1 and mini-batch size 100.

For **ReLU**, highest test accuracy (marked by white stars in the plot) is 96.3%, and is achieved by models with connectivity 0.06 (width 80) or 0.03 (width 160). For **Linear**, it is 92.7%, achieved at connectivity 0.13 (width 40). The color scheme is centered at the test accuracy value attained by the baseline model (approximately 90% in both cases), and its upper limit is set to the respective highest achieved value. Empty (white) cells correspond to invalid combinations of connectivity values. Note that the range on the horizontal axis is not equally spaced.

Figure 3.8: The MLP has one hidden layer, no biases, ReLU activation function, and NTK-style parametrization. It is trained on a subset of 2048 samples from the MNIST training set and tested on the full MNIST test set. The input is normalized with pixel mean and standard deviation as `(image - mean)/stdev`. We train for 300 epochs with vanilla SGD using Cross-Entropy loss and batch size 256. The learning rate was tuned separately for each width. The reported numbers are averages over 10 random seeds.

The number of weights in dense models is $(784+10)\cdot \mathit{width}$, while all sparse models have the same number of weights as the smallest dense model (width 8): 6352. The empirical approximation of the infinite-width kernel is computed on a dense MLP with width 10^4 at initialization.

Appendix B

Additional figures for ResNet-18 experiments

In this section we show additional plots for ResNet-18 experiments.

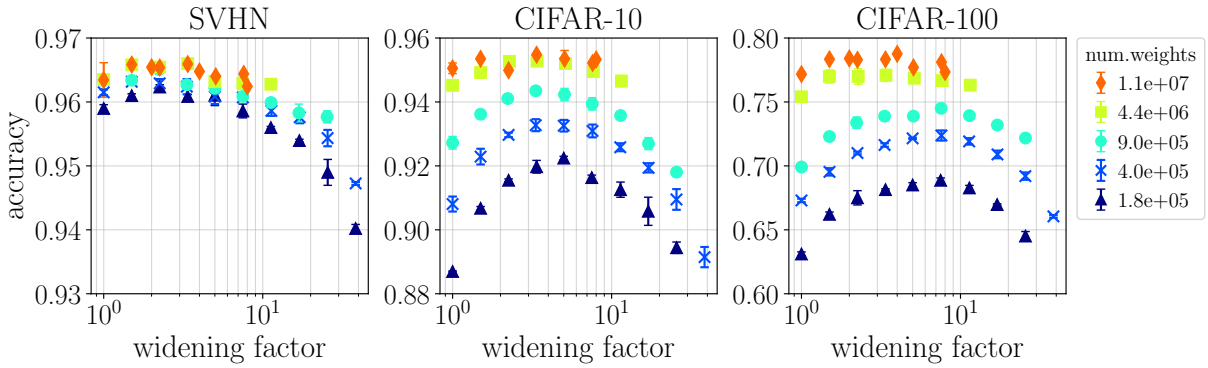


Figure B.1: Same data as in Figure 3.6a, but with error bars.

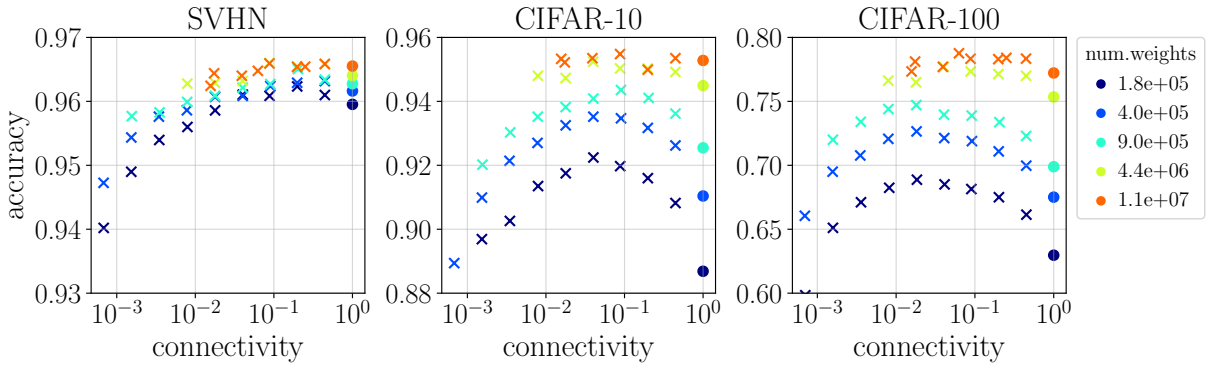


Figure B.2: Same data as in Figure 3.6a, but plotted as a function of network connectivity instead of width. Best test accuracy obtained by ResNet-18 models of different width (number of output channels of the first convolutional layer) and size (total number of weights, values indicated by marker color and shown in the legend). The rightmost data point of each color (filled circle) corresponds to the dense baseline model (connectivity 1), all other data points (crosses) correspond to its wider and sparser variants. For smaller models (up to 9.0e+5 weights), the performance peaks at similar connectivity values.

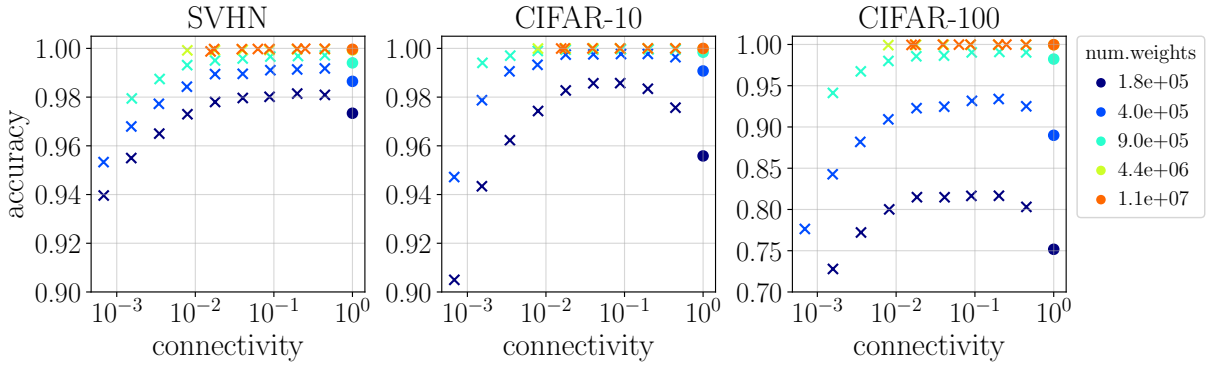


Figure B.3: Training accuracy from the same experiments as in Figures 3.6a and B.2. Larger models attain 100% training accuracy. For smaller models, the training accuracy shows a similar behavior as the test accuracy: It increases up to a certain connectivity and then deteriorates when the connectivity decreases further.

Appendix C

Model parametrization

Model parametrization includes two aspects: the forward pass, and the parameter initialization.

In **standard** parametrization, initial weight values for the ℓ-th layer are drawn independently from a normal distribution, $W_{ij}^{(\ell)} \sim \mathcal{N}(0, \sigma_\ell^2)$, with mean zero and standard deviation $\sigma_\ell = 1/\sqrt{\texttt{fan\_in}}$, where `fan_in` is the number of incoming connections per unit. In dense models with fully-connected layers, `fan_in` corresponds to the number of units in the preceding layer, $n_{\ell-1}$, and is the same for all n_ℓ units in layer ℓ. This initialization is also called *LeCun initialization*.

When we apply random static sparsity to such a model, the number of incoming connections per unit in layer ℓ can be less than $n_{\ell-1}$. We account for the sparse connectivity by setting the standard deviation to the *average* number of connections per unit, i.e. $\sigma = 1/\sqrt{p * \texttt{fan\_in}}$, where p is the connectivity.

In **NTK-style** parametrization, weights are initialized from a standard normal, $W_{ij}^{(\ell)} \sim \mathcal{N}(0, 1)$, while $1/\sqrt{\texttt{fan\_in}}$ is included as a prefactor in the forward pass (i.e., it multiplies the output of the given layer – see (3.1) for reference). For sparse models, we adjust the standard deviation in the normal distribution to be $\sigma = 1/\sqrt{p}$, but leave the forward-pass prefactor unchanged.

Appendix D

Sparsity distribution algorithm

The following code implements our algorithm for distributing sparsity over model layers. Figure 3.4 illustrates the procedure.

```
def get_ntf(num_to_freeze_tot, num_W, tensor_dims, lnames_sorted):
  """ Distribute the total number of weights to freeze over model layers.

    Parameters
      num_to_freeze_tot (int) - total number of weights to freeze.
      num_W (dict) - layer names (keys) and number of weights in layer (vals).
      tensor_dims (dict) - layer names (keys) and the dimensions of layer tensor
↪   (vals).
      lnames_sorted (list of str) - layer names, sorted by magnitude in descending
↪   order.

    Returns
      num_to_freeze (list of int) - number of weights to freeze per layer, order
↪   corresponding to lnames_sorted.
  """

  num_layers = len(lnames_sorted)
  num_to_freeze = np.zeros(num_layers, dtype=int) # init

  # list of num. weights in layer, in sorted order (largest first)
  num_W_sorted_list = [num_W[lname] for lname in lnames_sorted]

  # compute  num. weights differences between layers
  num_W_diffs = np.diff(num_W_sorted_list)
  num_W_diffs = [abs(d) for d in num_W_diffs]
```

```
# auxiliary vector for the following dot product to compute the bins
aux_vect = np.arange( 1,len(num_W_diffs)+1 )

# the bins of the staggered sparsification: array of max. num. of weights that can be
↪  frozen within the given layer before the next-smaller layer gets involved into
↪  sparsification
ntf_lims = [np.dot(aux_vect[:k], num_W_diffs[:k]) for k in range(1,num_layers)]

# find in which bin num_to_freeze_tot falls - this gives the number of layers to
↪  sparsify
lim_val, lim_ind = find_ge(ntf_lims, num_to_freeze_tot)
num_layers_to_sparsify = lim_ind+1

# base fill: chunks of num. weights that are frozen in each involved layer until all
↪  involved layers have equal num. weights remaining
base_fill = [sum(num_W_diffs[lind:lim_ind]) for lind in range(lim_ind)]
base_fill.append(0)

# the rest is distributed evenly over all layers involved
rest_tot = num_to_freeze_tot-sum(base_fill)
rest = int(np.floor(rest_tot/num_layers_to_sparsify))
num_to_freeze[:num_layers_to_sparsify] = np.array(base_fill)+rest

# first layer gets the few additional frozen weights when rest_tot is not evenly
↪  divisible by num_layers_to_sparsify
rest_mismatch = rest_tot - rest*num_layers_to_sparsify
num_to_freeze[0]+= rest_mismatch

assert sum(num_to_freeze)==num_to_freeze_tot

return num_to_freeze
```

Appendix E

Sparsity distribution in convolutional layers

The weight tensor of a convolutional layer has 4 dimensions: input, output, kernel width, kernel height. As discussed in section 3.4.3, when reducing network connectivity, we remove weights randomly across all dimensions of a weight tensor. A reasonable alternative for convolutional layers is to remove weights in the input and output dimensions only, leaving the kernels themselves unchanged. We test this approach on ResNet-18 and find that it leads to very similar results in general. Specifically for smaller models, removing weights along all tensor dimensions results in better performance. For the smallest networks (base widths 8 and 12) and small connectivity, we observed a gap up to 3% in test accuracy on both CIFAR data sets. Figure E.1 shows results for on CIFAR-100, where the effect is more pronounced, for models with base widths 8, 12 and 18 (1.8e5, 4.0e5 and 9.0e5 weights, respectively). Each experiment was repeated for 10 random seeds.

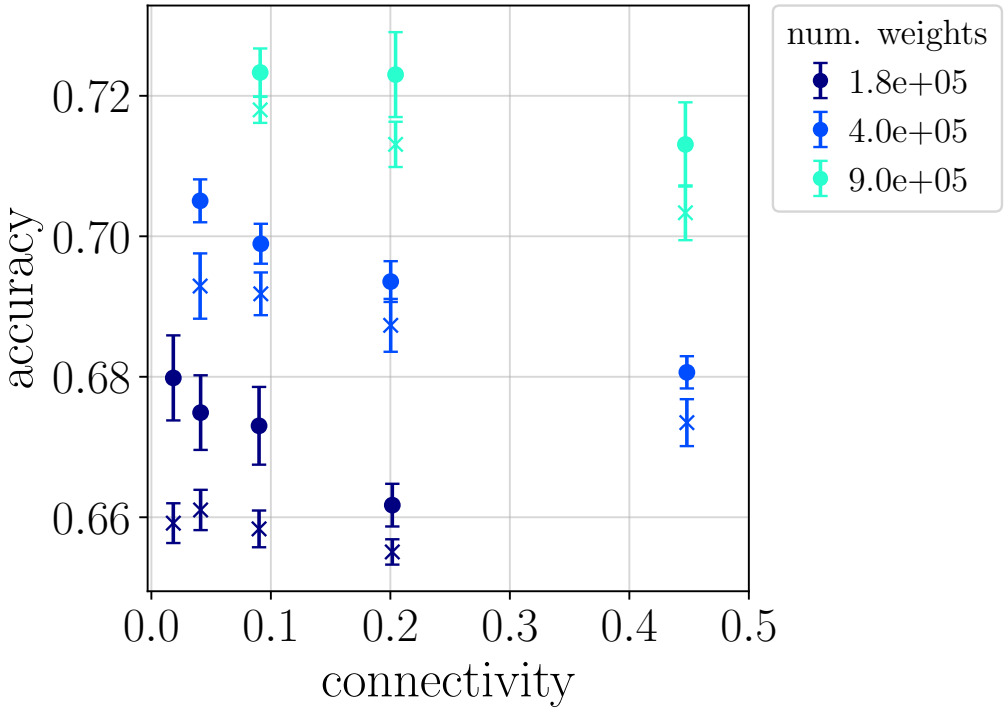


Figure E.1: Test accuracy achieved by ResNet-18 models on CIFAR-100, comparing performance of sparse wide models of different size (number of weights indicated by color and printed in the legend) given sparsity distribution in the convolutional layers along all layer dimensions (filled circle) versus along input/output dimensions only (cross).

Appendix F

Sparsity distribution in ResNet-18

On a coarse level, the ResNet-18 architecture is as follows: one convolutional layer, followed by four modules, followed by one fully-connected layer; each module comprises two blocks, and each block contains two convolutional layers. The number of output channels in the first convolutional layer is the same for the first module, and the ratio of output channel numbers in the subsequent modules is $1 : 2 : 4 : 8$ – we do not change this ratio in our experiments. When building a family of ResNet-18 models, we vary the number of output channels of the first convolutional layer, and refer to this as *the width* of the model, while the widths of all subsequent layers are set according to the mentioned ratio.

When reducing the connectivity of a ResNet-18 model, we remove weights from each layer according to layer size. More precisely, we first remove weights from the layer with the largest number of weights until it reaches the size of the next-smaller layer. We then proceed with removing weights from these two layers equally, and continue this procedure until the targeted total number of weights in the network is achieved.

Figures F.1 shows layer-wise sparsity distribution in ResNet-18 with 1.8e5 weights and various widths as an example.

Figure F.1: Layer-wise sparsity distribution in ResNet-18 of various widths with 1.8e5 weights, for the CIFAR-100 dataset.