universität
innsbruck

Institute of Theoretical Physics
University of Innsbruck

Master-Thesis

Artificial Neural Networks as Function Approximators in Quantum Many-Body Physics

Jonas Rigo, BSc

supervised by
Univ.-Prof. Dr. Andreas Läuchli
and
Dr. Thomas Lang

October 8, 2018

Abstract

Over the last few years the field of Machine Learning had great success in employing Deep Learning techniques to efficiently represent highly complex multivariate functions. Thereby, guidelines were found that help to design Neural Networks specifically to tackle a certain task. A great issue in quantum many-body physics is finding efficient representations of quantum wave-functions. Thus, we investigate Artificial Neural Networks as variational approach to find the ground-state wave-function of the anti-ferromagnetic Heisenberg Hamiltonian. Goal of our investigation is to derive guidelines for how to build a suitable network for this particular task, similar to the guidelines known from Machine Learning. We restricted our investigation to small system sizes, that are solvable by Exact Diagonalization, but we aimed for a network that can yield computationally exact ground-state energies. In our tests we observed that increasing the depth of neural networks increases the expressiveness of a network. Networks that are composed from activation functions with a specific symmetry are more successful to represent functions that agree with this symmetry. Increasing the depth makes networks less biased towards this intrinsic symmetry. Though to have computationally exact energies, activation functions are required to be unbiased towards symmetry. Other aspects of interest are training success and required number of parameters. We found that training success can significantly be increased with activation functions that have at no point a vanishing gradient. Regrading number of parameters, increasing depth to a moderate number (slightly more than two) allows to drastically reduce network parameters. In conclusion we found that one must respect the symmetry of the wave-function when trying to approximate it. Further redundant neurons and functions with strong gradient allow to increase training success, while depth can help reducing the necessary number of parameters (to accomplish a certain precision in energy). Ideally the derived guidelines help constructing networks to tackle larger systems and different Hamiltonians.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eigereicht.

Innsbruck, am October 8, 2018 Jonas Rigo

Contents

1. Introduction

In the field of Condensed Matter Physics the eigenstates of a quantum many-body system Hamiltonian are of huge interest. The eigenstates of the Hamiltonian are elements of the associated Hilbert space, whose dimension grows exponentially with the number of described particles. Therefore, one needs an exponential number of parameters to describe a general state exactly. Due to the enormous cost of such a description, techniques were developed that allow for a more efficient description. One of the most successful techniques that were found, are Tensor-Network approaches [1, 2, 3]. But as all these techniques do, also Tensor-Networks suffer from limitations, specifically the entanglement problem. Thus, systems that involve strong long-range correlations are so far inaccessible, and new techniques have to be developed to study them.

In recent years Deep Learning (a subcategory of Machine Learning) [4, 5] has provided many scientific fields with a new way to efficiently process big amounts of data. Deep Learning techniques were applied with great success to problems like object recognition [6, 7], natural speech processing [8] or recommendation systems [9] and performance is ever pushed further by improving algorithms and developing new specialized hardware [10, 11]. The origins of the power of Deep Learning are not yet fully understood. Though the employed neural networks [12, 13, 14] have the ability to capture correlations in data and extract useful features from data samples. Thus, neural networks are able to derive general concepts from a small training data set, these concepts allow the network to correctly process a much larger unknown set of data. In comparison to that, it is a well-known property of neural networks, that they can approximate almost any function arbitrarily well [15]. So the idea came up to employ neural networks in a new technique to approximate the ground-state of quantum many-body Hamiltonians [16]. The neural network described in the article by Carleo *et al.* is a so called Restricted Boltzmann Machines, which does not belong to the category Deep Learning, but this idea led to using more techniques from Machine Learning for the purpose of finding eigenstates of Hamiltonians [17, 18, 19, 20].

Investigations of the Restricted Boltzmann Machines uncovered its relation to Tensor-Networks, which showed that it can represent some interesting physical states exactly, like e.g. the Toric Code State [21, 22, 23]. The relation to Tensor-Networks also showed that the Restricted Boltzmann Machines is fulfilling either a volume entanglement law or an area entanglement law, where it depends on the architecture of the machine which one it is. A different approach is provided by Deep Feed Forward Neural Networks, which are networks used in Deep Learning and have no Tensor-Network representation. The goal is to exploit the great properties of Deep Learning for the quantum many-body problem. As already implied Deep Neural Networks are merely function approximators and are thus limited, in the precision of the representation, only by the complexity of the wave-function in the given basis. Other than the Restricted Boltzmann Machines one has many choices in designing Deep Neural Networks. The field of Deep Learning provides some heuristic guidelines on how to build Deep Neural Networks for specific tasks [24], but there are no rigorous statements about how to design a neural network for an arbitrary given problem. Thus, with this work we aim to provide guidelines and test-results that allow one to design a suitable Neural Network to investigate his Hamiltonian of interest.

This work can roughly be divided in two parts. In the first part (Sec. 2 - Sec. 4) we introduce the specific physical system we investigate and introduce the theoretic background that is necessary to discuss our numeric results. The second part is dedicated to the presentation and discussion of numeric results. We begin the second part with systematically constructing simple and unbiased networks (Sec. 5). From their test results we draw conclusions about how we can design the first specific network. The results we get from the specific networks allow us to iterative improve the network design until we settle for a final design. Eventually we introduce in Sec. 7 the Restricted Boltzmann Machines, due to its significance in this field and to use it to improve the performance of the presented Deep Neural Networks. In section (Sec. 5.6) we state the compact guidelines, that we derived from our investigation.

2. Physical Background

In the early days of material science, one had to investigate the materials that were provided by nature. Then technology allowed to fabricate materials. Techniques to synthesize materials range from forming new compounds of elements to arranging small nanocrystals in large periodic arrays and beyond [25, 26]. Some of the so discovered materials showed effects that were not observed before and required new theories to be described. A famous example is *high temperature superconductivity* [27], which is related to strong magnetic fluctuations that are also appearing in *low dimensional materials*. A low dimensional material is for example a material where the constituent particles sense only other particles that lie in the same plane (two-dimensional case) or along a linear chain (one-dimensional case). Such materials can then be described by effective two-dimensional, or one-dimensional models. Especially one-dimensional magnets have been studied intensively [28, 29, 30].

In the following we consider a one-dimensional lattice of regularly arranged atoms with high ionization energy and a single outermost electron in an s-orbital. Thus, for N atoms on the lattice we have N electrons in the system, which is referred to as *half-filling*, as every s-orbital could fit a second electron, to a total of $2N$ electrons in the system. When the electrons have only little kinetic energy, the Coulomb interaction dominates the system behavior. The resulting insulating phase can be described by only considering the spin-1/2 degree of freedom of every electron and we associate one spin to every lattice site [31]. In the next part of this text we introduce a simple Hamiltonian that governs the behavior of neighboring spins.

2.1. The Quantum Heisenberg Hamiltonian

Quantum mechanical spins are described by the spin-operators

$$S^x, S^y, S^z \ , \tag{2.1}$$

which are fully determined by the commutation relation

$$\left[S^\alpha, S^\beta\right] = \mathrm{i}\,\epsilon_{\alpha,\beta,\gamma} S^\gamma \ . \tag{2.2}$$

A single spin-1/2 state is an element of the Hilbert space $\mathbb{H} = \mathbb{C}^2$, thus the spin-operators have to be 2×2 matrices that fulfil Eq. (2.2). The only matrices that do so are the *Pauli matrices* ($\sigma^\alpha,\ \alpha \in \{x, y, z\}$), hence the spin operators read [31]

$$S^\alpha = \frac{\hbar}{2}\sigma^\alpha \ , \tag{2.3}$$

Let us assume that $|m\rangle \in \mathbb{H}$ is an eigenstate of the z-component spin-operator S^z, then applying S^z yields

$$S^z \, |m\rangle = \frac{\hbar m}{2} \, |m\rangle \ \ , \tag{2.4}$$

with m being the so called *magnetic quantum number*. The magnetic quantum number m can be either $m = 1$ or $m = -1$, where 1 (-1) means that the spin points upwards (downwards). To connect a given $|m\rangle$ to the next closest eigenstate of S^z one can use the *creation- and annihilation-operator* [31]

$$S^\pm = \frac{1}{2}(S^x \pm \mathrm{i}\, S^y) \ , \tag{2.5}$$

which can perform two actions each in the case of spin-1/2

$$S^+ \, |{-1}\rangle = \frac{\hbar}{2} \, |1\rangle \, , \ \ S^+ \, |1\rangle = 0 \tag{2.6}$$

$$S^- \, |{-1}\rangle = 0, \ \ S^- \, |1\rangle = \frac{\hbar}{2} \, |{-1}\rangle \ \ . \tag{2.7}$$

Eq. (2.6) shows how S^+ connects the state $|{-1}\rangle$ to $|1\rangle$, but when m is already $m = 1$ the state is mapped to zero, S^- acts similar but instead of raising m it lowers it.

As we have now introduced everything we need to treat a single spin we proceed to treat multiple spins. To each of the N spins we want to describe corresponds a Hilbert space $\mathbb{H} = \mathbb{C}^2$, the individual Hilbert spaces can be combined to a many-particle Hilbert space of dimension $\mathcal{D} = 2^N$

$$\mathbb{H}^{\otimes N} = \bigotimes_{i}^{N} \mathbb{C}^2 \ . \tag{2.8}$$

Any single spin operator O can be generalized to act on the i^{th} spin in terms of the Hilbert space $\mathbb{H}^{\otimes N}$ [31]

$$O_i = \left(\bigotimes_{j=1}^{i-1} I \right) \otimes O \otimes \left(\bigotimes_{j=i+1}^{N} I \right) \ , \tag{2.9}$$

where the index i denotes on which Hilbert space O is defined and I is the identity on $\mathbb{H}$. In some cases, it is useful to have an operator that acts on all sites, which we can obtain by summing over all O_i

$$O_{tot} = \sum_{i=1}^{N} O_i \ . \tag{2.10}$$

With Eq. (2.10) we can construct the total z-component spin operator

$$S^z_{tot} = \sum_{i=1}^{N} S^z_i \ . \tag{2.11}$$

To find the eigenbasis of S^z_{tot} we start building an eigenstate of S^z_{tot} by combining eigenstates $|m_i\rangle$ of S^z_i to

$$|(m_1, m_2, ..., m_N)\rangle = \bigotimes_{i=1}^{N} |m_i\rangle \ . \tag{2.12}$$

In Eq. (2.12) the set $\{m_i\}$ defines the whole eigenstate, thus to fully characterize any eigestate of S^z_{tot} we can use a vector $\mathbf{v} \in \{-1, 1\}^N$. This leads to the short hand notation for S^z_{tot} eigenstates and eigenvalues [31]

$$S^z_{tot} |\mathbf{v}\rangle = S^z_{tot} \bigotimes_{i=1}^{N} |v_i\rangle = \sum_i \frac{v_i \hbar}{2} |\mathbf{v}\rangle = \frac{m_{\mathbf{v}} \hbar}{2} |\mathbf{v}\rangle \ , \tag{2.13}$$

where $m_{\mathbf{v}}$ is the so called *magnetization* and $|\mathbf{v}\rangle$ is an eigenstate of S^z_{tot}. We denote the configuration space that corresponds to the eigenbasis of S^z_{tot} with

$$\mathcal{T} = \{-1, 1\}^N \ . \tag{2.14}$$

In Eq. (2.13) one can see that by reordering the entries in a vector $\mathbf{v}$ we do not change the eigenvalue, thus the eigenvalues of S^z_{tot} are degenerate with a degeneracy of [32]

$$d_{m_{\mathbf{v}}} = \binom{N}{|m_{\mathbf{v}}| + N/2} . \tag{2.15}$$

Let us now introduce the *anti-ferromagnetic Heisenberg Hamiltonian* [33]

$$\mathcal{H}_{ms} = J \sum_{\langle i,j \rangle} \mathbf{S}_i \cdot \mathbf{S}_j, \;\; J > 0 \;, \tag{2.16}$$

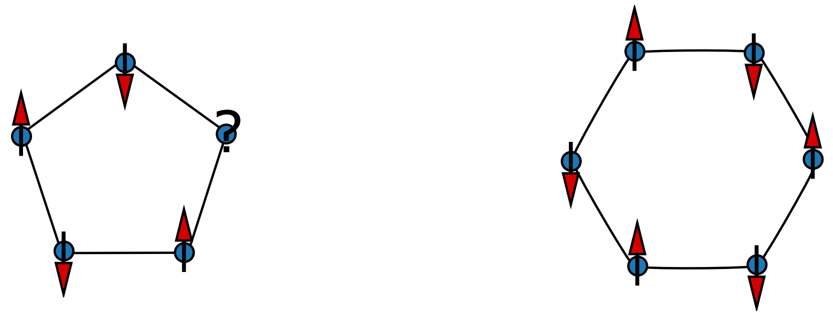
with $\mathbf{S}_i = (S^x_i, S^y_i, S^z_i)^T$. In the following we consider a spin chain with periodic boundary conditions, in other words Hamiltonian $\mathcal{H}_{ms}$ describes a ring of spins, as it is shown in Fig. 2.1. The product of two spin operators in Eq. (2.16) shows that $\mathcal{H}_{ms}$ governs the interaction of neighboring spins on the ring. We can rewrite $\mathcal{H}_{ms}$ in terms of the creation- and annihilation-operators to obtain a more convenient form of Eq. (2.16) [31]

$$\mathcal{H}_{ms} = \sum_{\langle i,j \rangle} J \left[\frac{1}{2} \left(S^+_i S^-_j + S^-_i S^+_j \right) + S^z_i S^z_j \right] . \tag{2.17}$$

When we check the commutation relation of S^z_{tot} and $\mathcal{H}_{ms}$ we find that they commute

$$[\mathcal{H}_{ms}, S^z_{tot}] = 0 \;, \tag{2.18}$$

this means that the magnetization is a good quantum number. Let us now clarify what *good quantum number* means. Take an eigenvector of the total z-component



(a) Spin-ring with five sites **(b)** Spin-ring with six sites

Figure 2.1: In there here displayed figures we show two types of spin rings. Every spin-ring consists of sites and bonds. The sites correspond to the spin-1/2 particles and the bonds indicate interactions. In Fig. 2.1a one can see the anti-ferromagnetic ordering for an odd site number, where we indicate the frustrated spin with a question mark. In Fig. 2.1b we have an even lattice numbering, which allows to order all spins opposed their neighbors.

spin operator and apply $\mathcal{H}_{ms}$ to it, then we obtain a superposition of vectors with the same magnetization as the original eigenvector [34]

$$\mathcal{H}_{ms} |\mathbf{v}\rangle = \sum_i \alpha_i |\mathbf{v}_i\rangle , \langle \mathbf{v}_i| S^z_{tot} |\mathbf{v}_i\rangle = \langle \mathbf{v}| S^z_{tot} |\mathbf{v}\rangle \ . \tag{2.19}$$

thus, a good quantum number is the eigenvalue of an observable that is conserved under application of the Hamiltonian. From this follows that the eigenstates of $\mathcal{H}_{ms}$ are a superposition of eigenvectors of S^z_{tot} with the same magnetization. Especially the subspace of states with zero magnetization is very useful, thus we denote it with

$$\mathcal{T}_0 = \{ |\mathbf{v}\rangle : S^z_{tot} |\mathbf{v}_i\rangle = 0 \} \ , \tag{2.20}$$

and it has the dimension

$$\dim(\mathcal{T}_0) = \frac{N!}{[(N/2)!]^2} \ . \tag{2.21}$$

Now we want to take a closer look at the Heisenberg Hamiltonian, the last term in Eq. (2.17) is an *Ising-type interaction*

$$\mathcal{H}_{ising} = J \sum_{\langle i,j \rangle} S^z_i S^z_j \ , \tag{2.22}$$

and we would like to minimize the energy of this part of the Hamiltonian. For an even number of sites, for example $N = 6$, we would do this as in Fig. 2.1b

$$\mathcal{H}_{ising} |(-1, 1, -1, 1, -1, 1)\rangle = -\frac{3J}{2} |(-1, 1, -1, 1, -1, 1)\rangle \ . \tag{2.23}$$

When we spin-flip the whole configuration in Eq. (2.23) we obtain a vector with the same magnetization and the same Ising term energy. For an odd number of sites, it is not completely obvious how one minimizes the Ising term. One can see in Fig. 2.1a that there is a site, that is left unset, with a question mark. The Ising term is minimal when all neighboring spins are opposed to each other, but for an odd number of sites we cannot satisfy this. One site can either be up or down and it has no effect on the energy

$$\mathcal{H}_{ising} |(-1, 1, -1, 1, -1)\rangle = -\frac{3J}{4} |(-1, 1, -1, 1, -1)\rangle \tag{2.24}$$

$$\mathcal{H}_{ising} |(-1, 1, -1, 1, 1)\rangle = -\frac{3J}{4} |(-1, 1, -1, 1, 1)\rangle \ , \tag{2.25}$$

Another consequence of this odd lattice numbering is that there is no $m_{\mathbf{v}} = 0$ and if you consider Eq. (2.24), the configurations that minimize the Ising term energy do not lie in the same magnetization subspace. Because of the odd lattice numbering, it can be shown that the ground-state is at least two-fold degenerate [35]. While for even N it can be shown that the ground-state is unique and lies in the space spanned by $\mathcal{T}_0$ and hence it holds [31]

$$S^z_{tot} \left|0\right\rangle = 0 \ . \tag{2.26}$$

In the following we only consider even lattice sizes, to avoid the ground-state degeneracy. Finally, we want to remark that the configurations presented in Eq. (2.23) and Eq. (2.24) do not minimize the whole Hamiltonian $\mathcal{H}_{ms}$, the creation- and annihilation-operator connect these simple states to other states, thus they are also not eigenstates.

2.2. Marshall Sign Transformation

In the previous subsection we state that we restrict our discussion to even lattice sizes, so Hamiltonian $\mathcal{H}_{ms}$ describes a spin ring with an even number of sites. This ring can be divided in two equally sized *sublattices* A and B with all even labeled sites being in A and all odd labeled sites being in B. This way we achieve a *bipartite* anti-ferromagnetic Hamiltonian to which we can apply Marshall's theorem [36]. Marshall's theorem states that one can rotate the spins on a sublattice such that for a given magnetization m the state with the lowest energy in this magnetization sector has only *positive definite coefficients* $q_\mathbf{v}$ in the S^z_{tot}-basis

$$\left|0\right\rangle = \sum_{\mathbf{v}\in\mathcal{T}:m_\mathbf{v}=m} q_\mathbf{v} \left|\mathbf{v}\right\rangle \,, \ q_\mathbf{v} \in (0,1] \ . \tag{2.27}$$

This result can also be achieved by *rotating the operators* acting on a sublattice, therefore we rotate the spin-operators around the z-axis on sublattice B [31]

$$\begin{aligned} i \in B : && (2.28)\\ S^+_i \mapsto -S^+_i, && (2.29)\\ S^-_i \mapsto -S^-_i, && (2.30)\\ S^z_i \mapsto S^z_i \ , && (2.31)\end{aligned}$$

which leads to Hamiltonian $\mathcal{H}_{ps}$

$$\mathcal{H}_{ps} = \sum_{\langle i,j\rangle} \left[-\frac{1}{2}\left(S^+_i S^-_j + S^-_i S^+_j\right) + S^z_i S^z_j \right] \ . \tag{ps}$$

Let us explain this result in a bit more detail as we assume that we know the ground-state $\left|0\right\rangle$ of $\mathcal{H}_{ps}$. We can express the ground-state in the eigenbasis of S^z_{tot}

$$\left|0\right\rangle = \sum_{\mathbf{v}\in\mathcal{T}} \left\langle \mathbf{v}|0\right\rangle \left|\mathbf{v}\right\rangle \ , \tag{2.32}$$

with $q_{\mathbf{v}} = \langle \mathbf{v}|0\rangle$ being the *coefficients*. Due to the lattice rotation Eq. (2.28) it holds that

$$\mathbf{v} \in \mathcal{T} : \langle \mathbf{v}|0\rangle \geq 0 \; , \tag{2.33}$$

where one will notice that the coefficients are not positive definite as we claim, but positive. We obtain the positive definite coefficients from Eq. (2.27) when we restrict the configuration set to $\mathcal{T}_0$

$$\mathbf{v} \in \mathcal{T}_0 : \langle \mathbf{v}|0\rangle > 0 \; . \tag{2.34}$$

The fact that for the ground-state of Hamiltonian $\mathcal{H}_{ps}$ positive definite parameters are enough, reduces the complexity to find the ground-state. But, much to the chagrin of Monte Carlo simulations [37], a transformation like Eq. (2.28) is not known for generic Hamiltonians. So, in general one has to deal with positive and negative, such as complex coefficients. To test if our variational Artificial Neural Network approach is also applicable to ground-states that have not exclusively positive coefficients we would like to discuss in the following the ground-state wave-function of Hamiltonian $\mathcal{H}_{ps}$ and $\mathcal{H}_{ms}$. Actually, the exact eigenstates and eigenenergies to Hamiltonian $\mathcal{H}_{ms}$ are known through the *Bethe ansatz* [38]. From there we learn that the ground-state is a real vector, thus Hamiltonian $\mathcal{H}_{ms}$ provides us with the desired ground-state problem that allows for more general evaluation of the quality of the neural network approach.

2.3. Variational Approach

Even when exploiting transformations as in Sec. 2.2 many-body Hamiltonians remain in general difficult to diagonalize [31]. Diagonalization of course refers to finding the solutions of the static *Schrödinger equation*

$$\mathcal{H} \left|i\right\rangle = \epsilon_i \left|i\right\rangle \; . \tag{2.35}$$

It can already be very interesting to know only the ground-state $|0\rangle$ of a Hamiltonian, or a good estimation of the ground-state energy. Physical intuition might allow someone to make a *variational ansatz* $|0_\theta\rangle$ for the ground-state that imitates the true ground-state, where parameters $\theta = \{\theta_1, \theta_2, ...\}$ characterize this trial state. The energy estimation for the trial state is the so-called *variational energy*

$$E = \frac{\langle 0_\theta| \mathcal{H} |0_\theta\rangle}{\langle 0_\theta|0_\theta\rangle} \; , \tag{2.36}$$

where dividing by the square-norm accounts for case when $|0_\theta\rangle$ is not normalized. For a Hamiltonian with a discrete set of eigenvectors $\{|i\rangle\}$ and eigenvalues $\{\epsilon_i\}$

it can be shown that equation Eq. (2.36) gives an upper bound for the actual ground-state energy $E \geq \epsilon_0$. So, when we minimize E

$$\frac{\partial E}{\partial \theta_1} = 0, \ \frac{\partial E}{\partial \theta_2} = 0, \dots \tag{2.37}$$

to determine the optimal values of θ we can never go below ϵ_0 and in the best case have exactly ϵ_0. When $E = \epsilon_0$ and the ground-state is non-degenerate it holds that $|0_\theta\rangle = |0\rangle$, for degenerate ground-states one might find any vector from the ϵ_0 subspace [39]. But in our discussion, we avoid this by restricting our self to even lattice sizes. In the case of $E > \epsilon_0$ the variational ansatz $|0_\theta\rangle$ contains also components from the eigenbasis of $\mathcal{H}$ that are orthogonal to $|0\rangle$. Assuming the eigenbasis is known and $|0_\theta\rangle$ is normalized, the overlap of $|0_\theta\rangle$ and $|0\rangle$ is

$$|q_0|^2 = |\langle 0_\theta | 0\rangle|^2 , \tag{2.38}$$

while the overlap of $|0_\theta\rangle$ and the rest of the eigenbasis $P_\perp(0) = I - |0\rangle\langle 0| \equiv |T\rangle\langle T|$ is

$$\mathrm{tr}(|0_\theta\rangle\langle 0_\theta| P_\perp(0)) = 1 - |q_0|^2 = \eta . \tag{2.39}$$

When the variational state is close to the exact state $\eta \ll \sqrt{\eta}$, which means for an observable O it holds that

$$|\langle 0_\theta| O |0_\theta\rangle - \langle 0| O |0\rangle| \approx \sqrt{\eta} |\langle 0_\theta| O |T\rangle| . \tag{2.40}$$

Therefore, the correlation function is more sensitive to deviations from the exact state than the ground-state energy [40].

In the case of Hamiltonian $\mathcal{H}_{ms}$ exact eigen-energies are provided by the *Bethe ansatz* [38] which is favorable in this case, since it allows to check the quality of our variational ansatz by evaluating the relative ground-state energy error

$$\Delta E = \frac{E - \epsilon_0}{\epsilon_0} . \tag{2.41}$$

Throughout the text we refer to ΔE as the *relative error* as it is the central quantity of our discussion. Though in our discussion we rely on the results of Exact Diagonalization [41] to get ϵ_0.

3. Inference of the Ground-State Quantum Wave-Function

The S^z_{tot} operator is a hermitian operator, hence it has a complete eigenbasis. In equation Eq. (2.13) we encode its eigenvectors with *configuration vectors* $\mathbf{v}_i \in \{-1, 1\}^N$.

Any state of the Hilbert space $\mathbb{H} = \bigotimes_{i=1}^{N} \mathbb{C}^2$ (with $\dim(\mathbb{H}) = \mathcal{D}$) can be expressed in the eigenbasis of S^z_{tot}. This includes of course also the ground-state of Hamiltonian $\mathcal{H}_{ms}$ which can be expressed as

$$|0\rangle = \sum_{i}^{\mathcal{D}} q_i \, |\mathbf{v}_i\rangle \ . \tag{3.1}$$

The amplitudes q_i in Eq. (3.1) can also be seen as a function of $\mathbf{v}_i$

$$q : \mathbf{v}_i \mapsto q(\mathbf{v}_i) \ . \tag{3.2}$$

In the following we discuss the *inference* of $q(\mathbf{v})$ from the point of view of *machine learning*. When we speak of *inference* in terms of machine learning we mean that we deduce a distribution form samples that are drawn from this distribution. Even though the process does not completely translate to finding the quantum wave-function we adopt the term *inferring* and use it in following for when we try to find the quantum ground-state wave-function.

3.1. Supervised Learning

A very prominent task in machine learning is *classification* [6]. Classification is a discriminate procedure where one assigns a label to a high dimensional vector as it does the function

$$f : \mathbb{R}^n \to \{1, ..., k\} \ . \tag{3.3}$$

A vector $\mathbf{x} \in \mathbb{R}^n$, which was assigned to a category $y \in \{1, ..., k\}$ must share some common *features* with other vectors that were assigned to the same category. An example for such a vector is an image and the features are color values of the single pixels. Now the discriminative task might be to decide whether this picture contains a cat or not. It is hard to design a deterministic program to accomplish this task, because there are way too many free parameters one has to consider. So what one does is to make a probabilistic ansatz $p_\theta(y|\mathbf{x})$ that tells how probable it is that a given vector $\mathbf{x}$ is in category y. To get the label, predicted by p_θ, one has simply to evaluate

$$f_\theta(\mathbf{x}) = \arg\max_{y \in \{1,...,k\}} p_\theta(y|\mathbf{x}) \ . \tag{3.4}$$

The distribution in our ansatz is characterized by a set of parameters θ. To determine these parameters, we take advantage of *training data*, which is a set of vectors that are already labeled

$$\mathcal{T} = \{(y_1, \mathbf{x}_1), (y_2, \mathbf{x}_2), ...\} \ . \tag{3.5}$$

Of course, one is more interested in correct predictions for vectors that are not already labeled. To cope with the lack of knowledge about f we assume a *prior* *distribution* for the deviation from the estimation by f_θ and for the parameters θ. For both distributions we assume a Gaussian distribution that is centered around zero. Hence for a sample of vectors and labels $\{(y, \mathbf{x})\} \sim \mathcal{T}$ from the training set we obtain

$$p(\{y\}|\{x\}, \theta, \beta) = \left(\frac{\beta}{2\pi}\right)^{|\{y\}|/2} \exp\left[-\frac{\beta}{2}\sum_i (f_\theta(\mathbf{x}_i) - y_i)^2\right] , \tag{3.6}$$

which reflects our assumption that the actual labels fluctuate in a Gaussian way around the estimated mean. For the parameters our assumptions yield

$$p(\theta|\alpha) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left(-\frac{\alpha}{2}\boldsymbol{\theta}\cdot\boldsymbol{\theta}\right) , \tag{3.7}$$

where the bold θ indicates that in this case we treat the set of parameters as vector. The combination of both distributions lets us obtain the *a posterior* distribution of the weights

$$p(\theta|\{\mathbf{x}\}, \{y\}, \alpha, \beta) \propto p(\{y\}|\{\mathbf{x}\}, \theta, \beta)p(\theta|\alpha) . \tag{3.8}$$

Of course, we are interested in the parameters that give the right prediction with the highest probability, in the field of Machine Learning this is referred to as *Maximum A Posterior Estimate* on which one can find greater detail in *Deep Learning* by Goodfellow *et al.* [4]. The process, that is referred to as *learning* (sometimes also *training*), consists of minimizing the *negative log likelihood*

$$E(\theta; \{\mathbf{x}\}) = -\log(p(\theta|\{\mathbf{x}\}, \{y\}, \alpha, \beta)) , \tag{3.9}$$

in terms of the parameters θ. The function E is also often referred to as *cost function*, since it is an objective function that provides a measure of accuracy for a given set of parameters θ. Optimizing the cost function might seem to be an easy task, but it is a very cumbersome procedure, because high performance on the training set does not induce good general predictions. To overcome these and other problems many strategies have been developed [42, 43] and in terms of finding the ground-state wave-function we treat this topic in greater detail later.

3.2. Unsupervised Learning

In the previous subsection we derive the cost function that must be optimized in order to infer a certain probability distribution with a probabilistic ansatz $p_\theta(y|\mathbf{x})$. Now we want to modify this procedure, such that we can use it to find the ground-state of a quantum Hamiltonian. The first approach one would take is to interpret the configurations $\mathbf{v} \in \{-1, 1\}^N$ as vectors to whom we want assign a label, which would be in this case the amplitudes $\{q\}$ of the quantum wave-function. This leaves us with the new training set

$$\mathcal{T} = \{(q_1, \mathbf{v}_1), (q_2, \mathbf{v}_2), ..., (q_\mathcal{D}, \mathbf{v}_\mathcal{D})\} \ , \tag{3.10}$$

which makes us realize that if we had all these amplitudes there would not be the problem of finding the ground-state in the first place. Thus, to find the right approach we take a step back and recall that in Sec. 3.1 we start out with an ansatz for the conditional probability distribution, which reads in the context of the ground-state

$$p_\theta(q|\mathbf{v}) \ . \tag{3.11}$$

The conditional probability distribution can be derived from the joint probability distribution in the following way

$$p_\theta(q|\mathbf{v}) = \frac{p_\theta(q, \mathbf{v})}{\sum_q p_\theta(q, \mathbf{v})} \ , \tag{3.12}$$

where $p_\theta(\mathbf{v}) = \sum_q p_\theta(q, \mathbf{v})$ is the probability of finding configuration $\mathbf{v}$. The nice thing about $p_\theta(\mathbf{v})$ is that we compensate our lack of knowledge by simply tracing over the unknown part. Thereby we obtain a quantity that is well known in quantum mechanics

$$p(\mathbf{v}) \propto |q(\mathbf{v})|^2 \ , \tag{3.13}$$

the probability of a configuration to occur, which is connected to the quantum mechanical amplitude through the square norm [44]. To infer the distribution $p(\mathbf{v})$ we do not need any *supervising* signal, like the category y in the supervised learning case, which is why such a procedure is called *unsupervised learning* (see Goodfellow *et al.* for greater detail [4]). Hence, the training set does not contain the supervising signal and becomes the already known set of configurations

$$\mathcal{T} = \{-1, 1\}^N \ . \tag{3.14}$$

Now we would like to find the prior and posterior distribution for our unsupervised learning problem. It is self-evident that the prior distribution is

$$p(\mathbf{v}|\theta) \propto |q_\theta(\mathbf{v})|^2 \ , \tag{3.15}$$

where $q_\theta(\mathbf{v})$ is a variational ansatz for the ground-state wave-function

$$|0_\theta\rangle = \sum_{\mathbf{v} \in \mathcal{T}} q_\theta(\mathbf{v}) \, |\mathbf{v}\rangle \ . \tag{3.16}$$

To find the posterior distribution we recall the variational energy form Sec. 2.3. The variational energy is minimal for the best possible parameters θ, just as the negative log likelihood, but we have two problems with the variational energy. First of all, it is not a probability distribution, but way more problematic is that the variational energy is *intractable* for arbitrary large system sizes. This means that in general we cannot evaluate the variational energy for $q(\mathcal{T})$, to overcome this problem one can make use of *Monte Carlo importance sampling*, which allows to generate a sequence of states $\{\mathbf{v}\}$ which is distributed according to p_θ, hence we say

$$\{\mathbf{v}\} \sim p_\theta \ . \tag{3.17}$$

Having such a sample $\{\mathbf{v}\}$ we can make *local estimations* of the variational energy instead of an exact evaluation [40]

$$\frac{\sum_{i,j} q_i^* q_j \langle \mathbf{v}_i | \mathcal{H} | \mathbf{v}_j \rangle}{\sum_i |q_i|^2} = \tag{3.18}$$

$$\sum_{i,j} p(\mathbf{v}_i) \langle \mathbf{v}_i | \mathcal{H} | \mathbf{v}_j \rangle \frac{q_j}{q_i} \approx \tag{3.19}$$

$$\sum_{\mathbf{v},\mathbf{v}' \in \{\mathbf{v}\}} p_\theta(\mathbf{v}') \langle \mathbf{v} | \mathcal{H} | \mathbf{v}' \rangle \frac{q_\theta(\mathbf{v}')}{q_\theta(\mathbf{v})} \equiv E_{loc}(\theta, \{\mathbf{v}\}) \ . \tag{3.20}$$

From Sec. 2.3 we know that the variational energy is an upper bound for the ground-state energy, thus we can cast it into a probability distribution for the quantum wave-function

$$p(\theta | \{\mathbf{v}\}, \beta) \propto \exp\left[-\beta E_{loc}(\theta, \{\mathbf{v}\})\right] \ . \tag{3.21}$$

This expression is the posterior distribution we were looking for, taking the logarithm yields the negative log likelihood

$$E(\theta; \{\mathbf{v}\}) = E_{loc}(\theta, \{\mathbf{v}\}) + \log(\beta) \ , \tag{3.22}$$

which is the desired *cost function*. Minimizing this cost function allows us to find the optimal variational parameters θ.

The unsupervised learning procedure was first used by Carleo *et al.* [16] to find a quantum ground-state. A striking difference to the supervised learning case is that samples from the training set $\mathcal{T}$ are generated using Monte Carlo importance sampling, instead drawing random samples. The importance sampling is somehow a way for the network to interact with its environment (the training set), which allows for further Machine Learning interpretation, e.g. Reinforcement Learning or Generative Learning [45, 4]. In this work we go not into these details, we restrict our discussion to system sizes that allow for an exact calculation of the variational energy for the full configuration set, which means that the cost function is the variational energy.

4. Feed Forward Neural Networks

To this day the brain is only partially understood and lots of effort is put into the investigation of this biological information processing unit. One can see this very impressively when considering that from a total of 108 Nobel prices for Medicine, 28 awarded achievements in the field of Neuroanatomy and Neurophysiology [46]. From the point of view of information processing the brain can be interpreted as a *network of neurons*, whereas every neuron is a processing unit by itself. In general, a computing model consists of three fundamental ingredients

$$computation = storage \; + \; transmission \; + \; processing. \tag{4.1}$$

In the case of the brain, which we see here as a *biological neural network*, the *transmission* is provided by interconnections between the neurons. These links feed input to the neurons to process, thereby the input from different links can be *weighted* individually before being processed. The weights associated to links are what resembles the *storage*, in terms of a neural network. So, after weighting the signal the neuron evaluates a *primitive function* and pass the processed signal on to the neurons to which it is connected to. This simple concept does not do justice to the vast complexity of a biological neural network, but it is a simplified model for which it can be shown that it can implement every logical function. Further the remarkable effectiveness of such networks is based on exploiting massive parallelism and redundancy, which allows to deal with unreliable single computing units. And even though all involved biologic processes are far slower than any electric information transmission and primitive processing, there are several problems that no computer can solve but the brain can [13], like completing the famous Turing test [47].
We take this as motivation to try and mimic the concept of biological neural networks with a mathematical formalism called Artificial Neural Network (ANN). The links between neurons are therefore replaced by channels which transmit an input v_i to the neuron. To each such channel we associate a weight w_i, just like to the link in the biological picture, which means that for a neuron with n channels the received signal is the weighted sum

$$w_1 v_1 + w_2 v_2 + \; ... \; + w_n v_n \; . \tag{4.2}$$

The neuron is, very much like in the biological case, the processing unit, thus a primitive function f evaluated on the input

$$f(w_1 v_1 + w_2 v_2 + \; ... \; + w_n v_n) \; . \tag{4.3}$$

In figure Fig. 4.1 we show a conceptual representation of this process. Concerning the function f there are no strict rules on what it should be. A very popular choice are functions that are $f(x \to \infty) = 0$ for large negative numbers and

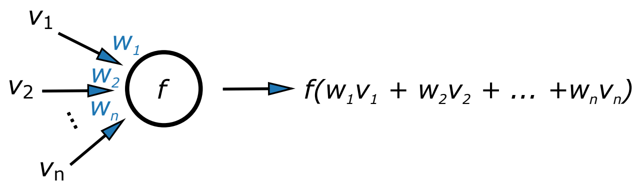


Figure 4.1: This figure shows a mathematical concept that mimics a biological neuron.

$f(x \to -\infty) = 1$ for large positive numbers and have a smooth monotonic transition in between, which should resemble a natural *activation potential*, from this convention we adopt the name *activation function* for f.

Now that we know how to implement a neuron we can consider the *topology* of a neural network. The topology determines from which other neurons a neuron receives its input and to which neurons it passes on its output. These connections allow us to implement different functions even when we use the same activation functions at the neurons. In Sec. 4.2 we dive deeper into the possibilities of expressing functions with neural networks. The topology is also what allows us to implement parallelism and redundancy in the network. In Fig. 4.2 one can see an example for an arbitrary topology with no actual purpose, thus we have left out the activation functions. This example lacks a certain structure and an order in which one calculates the node outputs best, thus we introduce in Sec. 4.1 a clear way to structure and evaluate an ANN. To quickly recap what determines an Artificial Neural Network we list some points

- links between artificial neurons define weighted sums,
- artificial neurons evaluate functions on a weighted sum,
- weights can be determined by using *learning algorithms*.

This last point regards the case when one uses an ANN as ansatz to approximate a target function q. Then the learning algorithm is an *optimization algorithm* that tries to find the best representation of q by adjusting the weights w_i associated to the networks links. Our target function q is the ground-state wave-function of Hamiltonian $\mathcal{H}_{ms}$ and $\mathcal{H}_{ps}$. The optimization algorithm used to approximate q with a neural network is subject of Sec. 4.3 and Sec. 4.4.

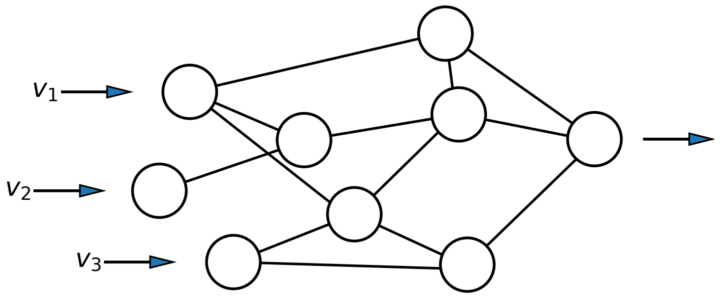


Figure 4.2: This drawing of a neural network is an example for an arbitrary topology, where we omit the functions on purpose, since they can also be chosen at will.

4.1. Forward-Propagation

The ANN that is shown in Fig. 4.2 has no structured neuron layout. The only rule that is followed in Fig. 4.2 is that the nodes, to which an arbitrary node feeds information, are always on the right-hand side of this node. This indicates that neurons that appear further right are evaluated after neurons that appear further left. We keep this convention but introduce the *layer* structure. The layer $L^{(i)}$ of an ANN consists of a certain number $n^{(i)}$ of neurons or *nodes*, this number defines the *layer-size*. All nodes contained in the layer $L^{(i)}$ are evaluated in parallel. In the special case of a *fully connected* neural network every neuron from layer $L^{(i)}$ is connected to every node in layer $L^{(i+1)}$, which is the layer further right form $L^{(i)}$ and therefore the one calculated later. Nodes in the same layer are not connected. In Eq. (4.2) we state that every connection corresponds to a weight $W_{jl}^{(i)}$ that multiplies with the input $a_j^{(i)}$, which is transmitted through the associated link. Thus, we can calculate the signal $z_j^{(i+1)}$ that the j^{th} node in layer $L^{(i+1)}$ receives from all the nodes in layer $L^{(i)}$

$$z_j^{(i+1)} = \sum_l^{n^{(i)}} w_{jl}^{(i+1)} a_l^{(i)} = \mathbf{w}_j^{(i+1)} \cdot \mathbf{a}^{(i)} \ . \tag{4.4}$$

We can combine all *weight vectors* $\mathbf{w}_j^{(i+1)}$ to a *weight matrix* $\mathbf{W}^{(i+1)} \in \mathbb{R}^{n^{(i+1)} \times n^{(i)}}$, that allows us to calculate all weighted signals at once (the following function is an affine function, but in the course of this work we refer to it as linear)

$$\mathbf{z}^{(i+1)} = \mathbf{W}^{(i+1)} \mathbf{a}^{(i)} + \mathbf{b}^{(i+1)} \ , \tag{4.5}$$

where we introduce the *bias* $\mathbf{b}^{(i+1)}$. The bias is like an additional neuron in layer $L^{(i)}$ which is only connected to neurons in layer $L^{(i+1)}$ and not in $L^{(i-1)}$. Other than usual neurons the bias needs no input signal to emit an output, it is always activated with the value 1. Thus, it is like adding an additional weight to the weighted sum in Eq. (4.2) that is unrelated to $\mathbf{a}^{(i)}$, which is what we do in Eq. (4.5). The next step of evaluating the neural network is computing, like in Eq. (4.3), the primitive function associated to a neuron. In general, every neuron in a layer $L^{(i)}$ can have a different activation function, but it makes things a lot easier when we assume that all neurons that are in the same layer share a common activation function $f^{(i)}$. Therefore, we introduce a writing convention to simplify the expressions

$$\mathbf{f}(\mathbf{x}) = \sum_{i}^{d} f(x_i)\mathbf{e}_i \ , \tag{4.6}$$

where $\{\mathbf{e}\}$ is the d dimensional standard basis. With the prerequisites above we can introduce what is called a *forward pass*. The forward pass takes the *activation* from layer $L^{(i)}$ and returns the activation from layer $L^{(i+1)}$ in the following way

$$\mathbf{a}^{(i)} = \mathbf{f}^{(i)}(\mathbf{z}^{(i)}) \tag{4.7}$$

$$\mathbf{z}^{(i+1)} = \mathbf{W}^{(i+1)}\mathbf{a}^{(i)} + \mathbf{b}^{(i+1)} \tag{4.8}$$

$$\mathbf{a}^{(i+1)} = \mathbf{f}^{(i+1)}(\mathbf{z}^{(i+1)}) \ . \tag{4.9}$$

Calculating the forward pass is also referred to *forward-propagation*, since it propagates the information through the network. To start the forward-propagation one has to initialize the first activation with an input, for this reason the first layer $L^{(0)}$ is called *input layer*. The activation of this layer is simply set to the input vector $\mathbf{v}$

$$\mathbf{a}^{(0)} = \mathbf{v} \ . \tag{4.10}$$

The signal in each following layer is then calculated based on this input, step by step. Layers $L^{(i>0)}$ are called *hidden layers*, besides the last layer. The last layer $L^{(\mathcal{L})}$, in a network of depth $\mathcal{L}$, is referred to as *output layer*. The sizes of all layers can be set freely, except for the input layer which must be of the same size as the input vector. At this point we would like to introduce some additional vocabulary for the building and treating an ANN. First of all, we use the term *architecture* to describe a network with a fixed number of layers $\mathcal{L}$ (depth) and a fixed set of functions $\{f^{(i)}\}_{i=1}^{\mathcal{L}}$ associated to the layers. The architecture tells nothing about the size of the single layers or the weights involved. This brings us to the next term we use throughout this text *parameters*, which we use to refer to the weights $W = \{\mathbf{W}^{(i)}\}_{i=1}^{\mathcal{L}}$ and the biases $b = \{b^{(i)}\}_{i=1}^{\mathcal{L}}$. The set containing all parameters is $\theta = \{W, b\}$, which resembles the notation that is used in Sec. 2.3.

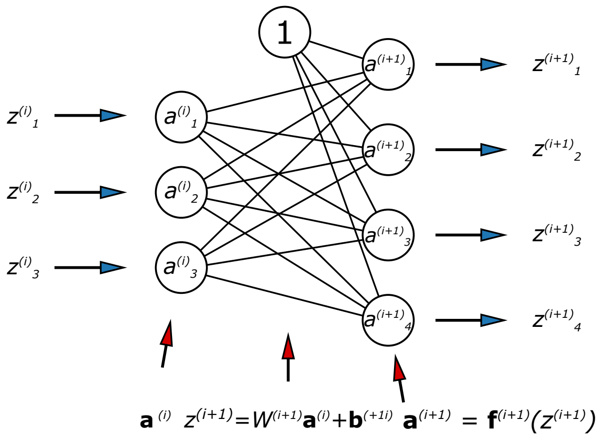


Figure 4.3: Here we illustrate a single forward-propagation step, as we define it in Eq. (4.7).

Having the depth $\mathcal{L}$, the set of functions $\{f^{(i)}\}_{i=1}^{\mathcal{L}}$ and parameters $\{W, b\}$ the neural network is fully defined. Basically, a neural network is a function that one can evaluate on an arbitrary input

$$h_{W,b} : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}, \; \mathbf{v} \mapsto h_{W,b}(\mathbf{v}) \; , \tag{4.11}$$

where $h_{W,b}$ is the *network function* and $d_{in} = n^{(0)}$, $d_{out} = n^{(\mathcal{L})}$. As an example we show how the network from Fig. 4.4 looks explicitly

$$h_{W,b}(\mathbf{v}) = \mathbf{f}^{(3)}(\mathbf{W}^{(3)}\mathbf{f}^{(2)}(\mathbf{W}^{(2)}\mathbf{f}^{(1)}(\mathbf{W}^{(1)}\mathbf{v} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) \; . \tag{4.12}$$

In the following we use neural networks as variational ansatz for the quantum ground-state wave-function, thus Eq. (3.16) becomes

$$|0_{W,b}\rangle = \sum_{\mathbf{v}\in\mathcal{T}} h_{W,b}(\mathbf{v}) \, |\mathbf{v}\rangle \;\; , \tag{4.13}$$

and we have therefore $n^{(0)} = N$ and $n^{(\mathcal{L})} = 1$.

4.2. Approximation Capabilities

In the previous section we introduce the mathematical structure of an ANN. Here our aim is to gain some qualitative understanding on how an ANN is able to approximate a function. With this we still cannot tell which the best ANN is to try as an ansatz for a specific function, but it is a step towards that. First we recall that the function we want to approximate is of the form Eq. (3.2), which is defined on the edges of a hypercube $\{-1, 1\}^N$ in the $\mathbb{R}^N$. The assumption we make next is not crucial for the question whether we can approximate a function $q(\mathbf{v})$ or not, it is rather a convenient move we make to ease the understanding. We assume that q is defined on the domain $[-1,1]^N$

$$q : [-1,1]^N \to [-1,1], \mathbf{v} \mapsto q(\mathbf{v}) \tag{4.14}$$

and moreover, to account for the normalization, q shall be an element of the *square integrable function space* $\mathcal{L}^2([-1,1]^N)$. When one fixes the architecture of a neural network, it is still unclear how many parameters the single layers should have. Assuming a certain configuration yields an approximation and one would like to improve that by increasing the parameter number. In such a case it is neither clear how many layers must be made larger nor to which extent. To resolve this ambiguity, for this discussion we only consider neural networks with two layers, where we omit the activation on the output layer. Such a network has the form

$$h_{W,b}(\mathbf{v}) = \sum_{i}^{n^{(1)}} w_i^{(2)} f^{(1)}(\mathbf{w}_i^{(1)} \cdot \mathbf{v} + b_i^{(1)}) \tag{4.15}$$

and is a function $h_{W,b}(\mathbf{v}) : \mathbb{R}^N \to \mathbb{C}$, when we allow complex output weights $w_i^{(2)} \in \mathbb{C}$. Before making statements about the expressivity of $h_{W,b}$ we need to be able to measure the accuracy of the approximation. The distance of $h_{W,b}$ to q, for a given environment measure μ, is measured with the $\mathcal{L}^2$ norm [15]

$$\rho_{2,\mu}(h_{W,b}, q) = \left[\int_{[-1,1]^N} |h_{W,b}(\mathbf{v}) - q(\mathbf{v})|^2 \,\mathrm{d}\mu(\mathbf{v})\right]^{1/2} . \tag{4.16}$$

This is certainly not the only way to do so, but it resembles the *mean squared error* that is often used in machine learning tasks [4]. To know if $h_{W,b}$ can approximate q we have to check whether q and $f^{(1)}$ fulfill certain criteria. The criteria q has to

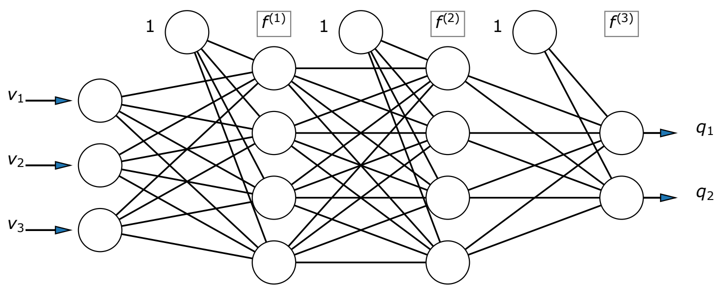


Figure 4.4: This figure shows what Eq. (4.12) describes. We have an activated layer that passes the activation on to the next layer and there we calculate the new activation.

meet, such that certain $f^{(1)}$ are able to approximate it, can be found in App. A. In our case these criteria do not matter to a certain extent, as q is actually discrete, and we only imagine a continuation on $\mathbb{R}^N$ that can meet any smoothness criteria (e.g. q can be interpreted as a smooth surface in N dimensional space). Given this it follows from App. A , according to Hornik *et al.* [15], that for $f^{(1)}$ being either

- *unbounded and non-constant* (for when q is square integrable) or
- *continuously differentiable, bounded and non-constant* (for when q is continuously differentiable)

there is for every $\epsilon > 0$ a number of neurons $n^{(1)}$ such that

$$\rho_{2,\mu}(h_{W,b}, q) < \epsilon \; . \tag{4.17}$$

This means that as long as $f^{(1)}$ meets the stated conditions we can find the right layer-size, such that we get the precision we please [15]. The number of required neurons and required training time depends on the chosen function $f^{(1)}$ [48], a good choice of $f^{(1)}$ can decrease the required layer-sizes compared to an arbitrary function. In general one can use any $f^{(1)}$, that meets conditions 4.2 and for this reason we refer to neural networks as *universal approximators.*

Another, but more instructive, approach to the same result can be obtained when we make the transition from the neural network as sum, to a representation as integral

$$h_{W,b}(\mathbf{v}) = \sum_{i}^{n^{(1)}} w_i^{(2)} f^{(1)}(\mathbf{w}_i^{(1)} \cdot \mathbf{v} + b_i^{(1)}) \tag{4.18}$$

$$\to \; h_{W,b}(\mathbf{v}) = \int T(\mathbf{w}, b) f^{(1)}(\mathbf{w} \cdot \mathbf{v} + b) \, \mathrm{d}\mathbf{w} \, \mathrm{d}b \; . \tag{4.19}$$

For this *integral representation* (as adopted from Sonoda *et al.* [49]) we replace the output layer weights with the density $T : \mathbb{R}^N \times \mathbb{R} \to \mathbb{C}$ and integrate over the first layer parameters $(\mathbf{w}, b) \in \mathbb{R}^N \times \mathbb{R}$, such that T is a continuous version of the output parameters. In this context the activation function $f^{(1)}$ can be seen as a *kernel.* When we know a function g, such that the pair $(f^{(1)}, g)$ is *admissible* (for the definition of *admissability* we refer to App. A) we can calculate, for the function q, the transformation

$$\left(\mathcal{R}^{\dagger}{}_{g} q\right)[\mathbf{w}, b] = \int q(\mathbf{v}) \overline{g(\mathbf{w} \cdot \mathbf{v} + b)} \, \mathrm{d}\mathbf{v} \; , \tag{4.20}$$

which is referred to as *Ridgelet transform* [49]. Note that the bar $\overline{\;\cdot\;}$ in Eq. (4.20) denotes complex conjugation. Here we consider only the case when we are given the right function g, such that we are able to calculate the density $T = \mathcal{R}^{\dagger}{}_{g} q$ which then allows us to obtain q again

$$\left(\mathcal{R}_{f^{(1)}}\mathcal{R}^{\dagger}{}_{g} q\right)[\mathbf{v}] = \int T(\mathbf{w}, b) f^{(1)}(\mathbf{w}\cdot\mathbf{v} + b)\,\mathrm{d}\mathbf{w}\,\mathrm{d}b \propto q(\mathbf{v}) \,, \tag{4.21}$$

using the so called *dual Ridgelet transform.* As one can find in App. A this *reconstruction* of q is possible for a broad class of functions $f^{(1)}$ which shows again, but in a slightly different way, that neural networks are universal approximates. The actually interesting thing is that, if the function g is known, we could construct an exact representation of the function $q \in \mathcal{L}^2([-1,1]^N)$ with the integral representation of the neural network. One can compare this with the *Fourier transform* where $f^{(1)}(\mathbf{w}\cdot\mathbf{v}) = g(\mathbf{w}\cdot\mathbf{v}) = \mathrm{e}^{-\,\mathrm{i}\,\mathbf{w}\cdot\mathbf{v}}$, which allows us to calculate the density $T(\mathbf{w})$ that can be used to reconstruct the function q exactly. But numerically speaking this is very unhandy, thus the idea of *discrete Fourier transformations* comes up [32]. The idea behind discrete Fourier transformations is to discretize $T(\mathbf{w})$ in a way that the reconstruction of q yields a good approximation, using the finite information contained in T. And here we meet again our original neural network that follows the same idea of discretizing an integral representation to obtain a good approximation of the exact function. Finally, we come again to realize that a neural network can approximate a function q, when we are not limited in the number of nodes and the activation function fulfills some criteria. The new insight we gain is that a neural network representation is simply a very general discretized integral representation of a function. Further we open our sight for a different kind of activation functions, the sine and cosine functions from the Fourier transform. As the discrete Fourier transform is also a type of neural network, it might be

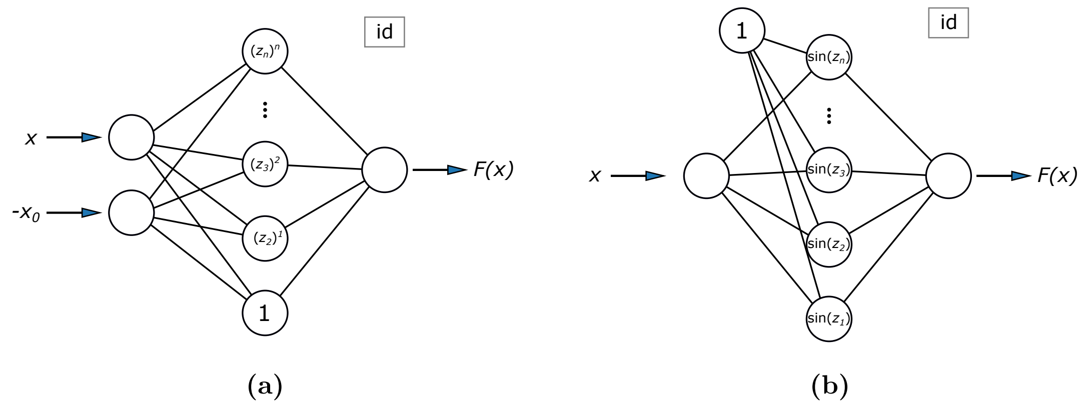


Figure 4.5: In Fig. 4.5a one can see the realization of a two-layer *Taylor network*, which is meant to resemble a Taylor expansion. In a similar fashion we have a *Fourier Network* in Fig. 4.5b that resembles discrete Fourier transform.

advantageous to use a cosine or sine activation function in one or more layers, when dealing with repetitive patterns in q. In Fig. 4.5b we show an example of how a Fourier network might look like

$$F(x) = \sum_{i}^{n} w_i^{(2)} \sin(w_i^{(1)} x + b_i^{(1)}) \;, \tag{4.22}$$

or in a similar fashion we can transform a *Taylor expansion* into a network like Fig. 4.5a [13]

$$F(x) = \sum_{i}^{n} w_i^{(2)} (x - x_0)^i \;, \tag{4.23}$$

with $w_i^{(1)} = 1$.

This brings us to the question how many layers a network should have. So far, we know that one layer is enough for our desired approximation and stacking another on top of it, or as many as one wants, changes nothing about the generality [15]. The idea of stacking layers, like we do in the example Fig. 4.4, makes the difference between a simple discretized integral representation and the more powerful neural networks. But this concept only makes sense when we achieve a desired approximation while using less parameters with more layers, than we do with less layers. We can also ask the other way around: Is there a function, which an $\mathcal{L}$-layer network can approximate with $|\theta| = C$ parameters and an $(\mathcal{L}-1)$-layer network cannot approximate as well, even if it is allowed to use much more parameters than C? Eldan *et al.* present in [50] a class of functions for which this is the case and we try to depict some of their findings with an example. Consider q, only for this purpose, to be a radial function. Further consider that, instead of a two-layer network as before, we have now a three-layer network

$$h_{W,b} = \sum_{i}^{n^{(2)}} w_i^{(3)} f^{(2)} \left[\sum_{j}^{n^{(1)}} w_{i,j}^{(2)} f^{(1)}(\mathbf{w}_j^{(1)} \cdot \mathbf{v} + b_j^{(1)}) + b_i^{(2)} \right] \;. \tag{4.24}$$

A radial function can be expressed using a norm $|\cdot|$ and a function $r : \mathbb{R} \to \mathbb{R}$, such that

$$q : \mathbb{R}^N \to \mathbb{R}, \mathbf{v} \mapsto r(|\mathbf{v}|) \;. \tag{4.25}$$

One can easily imagine how we can express q with a three-layer network as in Eq. (4.24). The first to layers of Eq. (4.24) form a two-layer network and thus a universal approximator. Using this part of the network one can construct a superposition of neurons that expresses the norm $|\cdot|$, then taking this as the input for the next approximation, which is the output layer. The output layer can then make use of the provided superposition to express r, to a certain degree of accuracy.

In terms of a two-layer network, it is not clear how this procedure translates, and it appears to be much harder. This hand waving argument can be cast into a strict mathematical theorem, which can be found in App. A. The theorem states that there is a class of functions expressible as a three-layer network, that has polynomial many neurons, with respect to the input dimension N. A two-layer network, composed of at most exponentially many neurons with respect to N, is lower bounded in its ability to approximate (regarding the $\mathcal{L}^2$ norm) the three-layer network. This is an explicit example for when one can save parameters by increasing the depth of the network.

So far, we know that for any function there is a number of neurons that allows us to approximate it with a certain precision. Now we realize that we can reduce this number in some cases by using a deeper network. But deeper networks can also cause difficulties and are not generally better than shallow networks. Some of the problems, that depth causes require a lot of effort to be resolved but need to be overcome in order to make deep networks feasible [42, 43]. In the following we discuss the training algorithm for neural networks and in the course of this we stumble upon some of these difficulties.

4.3. Gradient Descent

Before we dive deep into the specific techniques and problems, that arise in the procedure of training an Artificial Neural Network, we recall the basics of *gradient-based optimization*. The training of an ANN is merely a special case of gradient-based optimization, that aims to give us the argument for which a cost function f is either maximal or minimal, but we settle for the case of desired minimum

$$\mathbf{x}^* = \arg\min f(\mathbf{x}) \; . \tag{4.26}$$

To get an intuition for the underlying concept of gradient-based optimization assume a one dimensional differentiable function $f : \mathbb{R} \to \mathbb{R} : x \mapsto f(x)$. For a small variation $\eta > 0$ around an argument x we can approximate f with a linear function in η

$$f(x + \eta) \approx f(x) + \eta f'(x) \; , \tag{4.27}$$

where $f' = \mathrm{d}f/\,\mathrm{d}x$ is the derivative of the function. Now suppose that we are looking to minimize f, then we know by the derivative that for small enough η

$$f(x - \eta f'(x)) < f(x) \; , \tag{4.28}$$

simply because

$$f(x - \eta f'(x)) \leq f(x) - \eta(f'(x))^2 < f(x) \; . \tag{4.29}$$

The intuition behind Eq. (4.29) is that the derivative provides a *direction* in which we can move from x to minimize f. But note that the derivative does not correspond to the ideal *step size*, the amplitude of the derivative is not correlated to its range of validity. In general, only small step sizes lead to a minimization, hence it is enough to calculate $\operatorname{sign}(f'(x))$ and thus Eq. (4.29) becomes

$$f(x - \eta \operatorname{sign} f'(x)) \leq f(x) - \eta \operatorname{sign} f'(x) < f(x) \; . \tag{4.30}$$

With this in mind we can imagine a simple optimization algorithm. The procedure we are about to state is called *gradient descent* and works as follows [4]

1. Calculate the sign of the derivative in x and then
2. update x with a small step with the opposite sign $x \leftarrow x - \eta \operatorname{sign} f'(x)$.

This procedure can be repeated until $f'(x) = 0$, which can either be the desired *global minimum* or a not desired *local minimum*, *maximum*, such as a *saddle point*. Our algorithm cannot distinguish between these cases. For a multidimensional function $f : \mathbb{R}^d \to \mathbb{R}$ we replace the derivative $\mathrm{d}/\,\mathrm{d}x$ with the gradient ∇, but the interpretation stays the same. In analogy to the one-dimensional case the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$ of the function f points towards the *steepest descent*. The steepest descent is the direction in which one has to move the argument in order to increase or decrease the value of f the fastest. Eventually, also in the multidimensional case we have a problem when $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$.

Our goal is to use gradient descent to minimize the cost function $E(\theta; \{\mathbf{v}\})$ from Sec. 3.2. The cost function E is essentially the variational energy, the crucial point is that the variational ansatz for the wave-function is a neural network

$$q_\theta(\mathbf{v}) \leftarrow h_{W,b}(\mathbf{v}) \; . \tag{4.31}$$

Thus, the cost function becomes

$$E(\theta; \{\mathbf{v}\}) \leftarrow E(W, b; \{\mathbf{v}\}) \; , \tag{4.32}$$

where $\{\mathbf{v}\} = \mathcal{T}$. To optimize E we cannot yet use the gradient descent algorithm as we introduce it, because it lacks a way to calculate the step size η. The step size η is in Machine Learning known as the *learning rate* and belongs to the class of *hyper-parameters*. Most commonly one employs adaptable hyper-parameters and here in particular we use the *Adagrad* algorithm to calculate η in every optimization step [51]. Since the value of η depends on the iteration step we denote the gradient in the t^{th} iteration with

$$g^W_{t,i} = \nabla_{W^{(i)}} E(W, b; \{\mathbf{v}\}) \tag{4.33}$$

$$g^b_{t,i} = \nabla_{b^{(i)}} E(W, b; \{\mathbf{v}\}) \; . \tag{4.34}$$

When we leave out the superscript we mean to do the same operation with both gradients. Adagrad updates the learning rate at every time step and for every parameter individually according to the square sum of previous gradients. We initialize this square sum with zeros $G_{0,i} = 0$ and use then the following update rule

$$G_{t,i} = G_{t-1,i} + g_{t,i} \circ g_{t,i};\,, \tag{4.35}$$

with $\circ$ denoting element-wise multiplication. Now we can calculate the parameter update

$$\mathbf{W}^{(i)} \leftarrow \mathbf{W}^{(i)} - \frac{\eta}{\sqrt{G^W_{t,i} + \epsilon}} \circ g^W_{t,i}\,, \tag{4.36}$$

where ϵ is a positive definite smoothing parameter. This method has the advantage to be a self-regulating update, which decreases monotonically. One has still to choose an initial η, but it loses its significance to the update the more the longer the training process runs.

When we employ Adagrad in the gradient descent algorithm, to minimize E, we begin by setting $G_{0,i} = 0$, $t = 1$ and $\{W, b\}$ are initialized with Gaussian distributed random values. Then we follow the steps of the algorithm for a fixed number of iterations:

1. Calculate for all layers ($0 < i \leq \mathcal{L}$) the gradient
$$\begin{aligned} g^W_{t,i} &= \nabla_{W^{(i)}} E(W, b; \{\mathbf{v}\}) \\ g^b_{t,i} &= \nabla_{b^{(i)}} E(W, b; \{\mathbf{v}\})\,. \end{aligned}$$

2. Calculate for all layers the square sum $G_{t,i} = G_{t-1,i} + g_{t,i} \circ g_{t,i}$.

3. Finally update all layer weights
$$\begin{aligned} \mathbf{W}^{(i)} &\leftarrow \mathbf{W}^{(i)} - \frac{\eta}{\sqrt{G^W_{t,i} + \epsilon}} \circ g^W_{t,i}, \\ \mathbf{b}^{(i)} &\leftarrow \mathbf{b}^{(i)} - \frac{\eta}{\sqrt{G^b_{t,i} + \epsilon}} \circ g^b_{t,i}, \\ t &\leftarrow t + 1\,. \end{aligned}$$

As far as the optimization process is concerned we have introduced everything crucial besides how the gradients Eq. (4.33) are calculated. This specific calculation is subject of the next section.

4.4. Back-Propagation

The gradient descent algorithm form Sec. 4.3 is not complete without the calculation of the gradient of the cost function. For Artificial Neural Network the calculation of the gradient is called *back-propagation*, which is a form of automatic differentiation algorithm presented first by Seppo Linnainmaa in his master's thesis [52]. In this section we introduce the back-propagation algorithm and discuss some of the flaws it has. But before coming to back-propagation we must discuss the cost function.

The basic idea of our discussion is to use a neural network $h_{W,b}$ as variational ansatz for a ground-state wave-function q. The quantum wave-function is in general complex functions [44]

$$q : \mathbb{R}^N \to \mathbb{C} : \mathbf{v} \mapsto q(\mathbf{v}) \ , \tag{4.37}$$

in Sec. 4.2 we account for this case by making the weights of the output layer complex. But in this scenario the output layer has no activation function, the output is linear. But one can work around this like we do in Fig. 4.6c, where we have a network with two output neurons $n^{(\mathcal{L})} = 2$ and add a layer with fixed weights $\mathbf{w}^{(\mathcal{L}+1)} = (1, \mathrm{i})$ to the output. This way complex and real part of the output is treated equally, and it has the same effect as having a complex output weight. A different approach is shown in Fig. 4.6b, the idea is the same with the difference that the additional layer is activated with $f^{(\mathcal{L}+1)} = \exp$. In this scenario the first activation of the output layer corresponds to the amplitude and the second activation corresponds to the phase of a complex number. In Sec. 2.1 we state that the ground-state wave-functions of $\mathcal{H}_{ps}$ and $\mathcal{H}_{ms}$, which we want to find, have only real amplitudes, thus we need only one output neuron as in Fig. 4.6a. But to generalize our results Fig. 4.6c is the right approach, as we do never investigate the impact of an exponential activation function (see Tab. 1 for

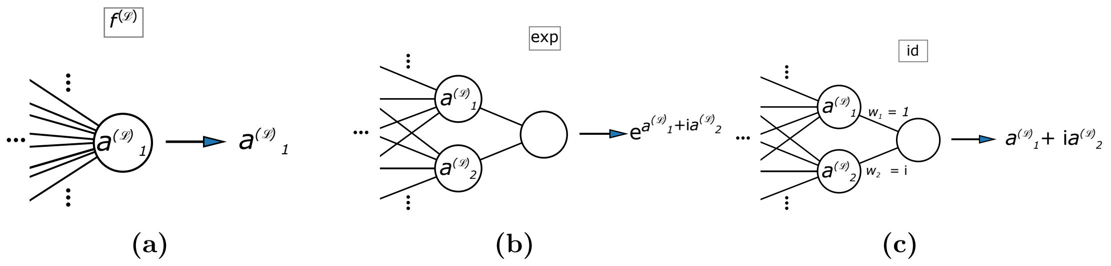


Figure 4.6: In this figure we show how to implement a real and complex output with a neural network. In Fig. 4.6a we have a one-dimensional real output. The configuration in Fig. 4.6b allows for a complex output, where the network provides amplitude and phase. In Fig. 4.6c the network has also a complex output, but the real and imaginary part of the output are taken directly from the network.

the list of activation functions we discuss). So to be clear the networks we treat in the following have $n^{(0)} = N$ and $n^{(\mathcal{L})} = 1$. As this is settled we can state the explicit form of the cost function Eq. (4.32)

$$E(W, b; \{\mathbf{v}\}) = \frac{\sum_{i,j} h_{W,b}(\mathbf{v}_i) h_{W,b}(\mathbf{v}_j) \langle \mathbf{v}_i | \mathcal{H} | \mathbf{v}_j \rangle}{\sum_i h_{W,b}(\mathbf{v}_i)^2} , \tag{4.38}$$

where one might notice that we omitted the complex conjugation, since the network $h_{W,b}$ has no imaginary output.

Recall that to do gradient descent with the cost function $E(W, b; \{\mathbf{v}\})$ we need to differentiate it with respect to the parameters of the network $h_{W,b}$. This derivative can be provided by the back-propagation algorithm, which is conceptually similar to the forward-propagation algorithm. If we recall the forward-propagation from Sec. 4.1 there is this central concept of calculating the activations of one layer and then passing them on to the next layer for processing. When we calculate the gradient of E, the procedure is very similar, but instead of starting from the first layer we start at the last layer. The cost function E can be partially differentiated with respect to the parameters of the last layer. This result can then be used, by employing the *chain rule*, to calculate the derivative with respect of the precedent layer. In this way, the gradient propagates through the network like the output does in the opposite direction. Strictly speaking this hand-waving explanation corresponds to the following algorithm.

1. Compute a forward pass $h_{W,b}(\{\mathbf{v}\})$.

2. For the last layer $i = \mathcal{L}$

$$\boldsymbol{\delta}^{(\mathcal{L})} = \frac{\partial E(W, b; \{\mathbf{v}\})}{\partial \mathbf{z}^{(\mathcal{L})}} \circ \mathbf{f}'(\mathbf{z}^{(\mathcal{L})}) ,$$

   where $\circ$ is an element-wise multiplication (see App. B for the explicit form).

3. For each layer $i = \mathcal{L} - 1, \mathcal{L} - 2, ..., 1$ calculate

$$\boldsymbol{\delta}^{(i)} = (\mathbf{W}^{(i)})^T \boldsymbol{\delta}^{(i+1)} \circ \mathbf{f}'(\mathbf{z}^{(i)}) . \tag{4.39}$$

4. Compute the desired derivatives

$$\begin{aligned} \nabla_{W^{(i)}} E(W, b; \{\mathbf{v}\}) &= \boldsymbol{\delta}^{(i+1)} (\mathbf{a}^{(i)})^T \\ \nabla_{b^{(i)}} E(W, b; \{\mathbf{v}\}) &= \boldsymbol{\delta}^{(i+1)}. \end{aligned}$$

Performing gradient descent with this way to calculate the gradients is often referred to as *training a neural network*. The configuration set $\{\mathbf{v}\} = \mathcal{T}$ is in our case the training set and when one calculates the gradient for the whole training set, not just a subset, it is called *batch gradient descent* [4].

Now let us take a moment to take a closer look to some of the flaws of the back-propagation algorithm. Besides the problems of gradient descent itself, the back-propagation algorithm has some more problems particularly because of the way how the gradient is calculated. The problems we face in this work are known under the name *vanishing and exploding gradient problem.* These problems arise at two points in equation Eq. (4.39). At first when we calculate

$$\boldsymbol{\delta}^{(i)} = (\mathbf{W}^{(i)})^T \boldsymbol{\delta}^{(i+1)} \circ \mathbf{f}'(\mathbf{z}^{(i)}) \tag{4.40}$$

we obtain the product of the $\mathcal{L} - i$ previous weight matrices. Before starting the training procedure, we initialize W and b randomly, thus we have only little information about the eigenvalues of $\{\mathbf{W}^{(i)}\}$. When we are unlucky and all eigenvalues smaller than one, then the gradient vanishes more for every layer. The other extreme case would be very large eigenvalues. In that case the gradient grows exponentially with every layer. Fortunately, this problem disappears during the training of the specific type of neural networks that we use here, since we do not retain any information about the previous gradient, after a full gradient update, and thus we do not have the possibility of exponential growth or decay of the gradient [53]. But we are not done with Eq. (4.39). A instance that actually affects us in this equation is that in every step we multiply the previous $\delta^{(i+1)}$ not only with the transposed weight matrix, but also with the derivative of the function $f^{(i)}$. To see why this might become a problem take the example of the activation function $f^{(\mathcal{L})} = \tanh$ in the last layer. In that case all the previous layers depend on the derivative of this function. But for large values the tanh is very well approximated by a constant, thus it has derivative nearly zero. As the updates for the previous layers are multiplied with this derivative they also vanish, and the network does not improve anymore by updates. This is a point that one has to consider when designing a neural network. A way around this is the use of the *rectified linear unit* (ReLU) [54]

$$\mathrm{ReLU} : \mathbb{R} \to [0, \infty) : x \mapsto \max\{0, x\}\ . \tag{4.41}$$

In Sec. 4.2 one can find that ReLU fulfills all the conditions to be an activation function that yields a universal approximator. The rectified linear unit is a non-linear function but acts as a linear function for strictly positive input. This leads to some interesting properties which go beyond the scope of this work. What is most important for us is that the derivative of ReLU shows the convenient behavior

$$\frac{\mathrm{d}}{\mathrm{d}x} \mathrm{ReLU}\,|_z = \begin{cases} 0, & \text{if } z < 0. \\ 1, & \text{otherwise.} \end{cases}\ , \tag{4.42}$$

of a not vanishing gradient.

In the following we use batch gradient descent to train neural networks with different architecture. In the course of this we can observe the vanishing gradient problem but also the advantage of the ReLU.

5. Towards a Computational Exact Ground-State Energy

Artificial Feed Forward Neural Networks were already used to find the ground-state of bosonic and fermionic many-particle Hamiltonians. These examples of usage had the purpose to show the practicability of ANNs as ground-state approximators, thus showed different kinds of ANNs and how they perform compared to other approximation techniques (like the Density Matrix Renormalization Group [2, 3]). For example, Cai *et al.* tested in [17] a four-layer neural network with fixed layer sizes in a supervised learning scheme (for a Hamiltonian with known ground-state, see Sec. 3.2 for greater detail) and tried different activation functions to improve performance. Saito *et al.* investigated in [18] Feed Forward Neural Networks and *Convolutional* Feed Forward Neural Network (see [4] for greater detail or Sec. 6.2 for a short explanation) in an unsupervised learning scheme, employing Monte Carlo importance sampling. They discussed the impact of depth on the approximation capability of their neural network models. In this work we take a similar approach and yet quite different. First of all, we treat only small system sizes for which we can calculate the variational energy exactly. The before mentioned discussions set performance benchmarks with a network and then tried to improve upon this benchmark, we search explicitly a neural network that achieves computationally exact ground-state energies for $\mathcal{H}_{ms}$ and $\mathcal{H}_{ps}$

$$\Delta E = \frac{E - \epsilon_0}{\epsilon_0} < 10^{-15} \,. \tag{5.1}$$

Therefore, we systematically create and test ANNs to find how we can achieve this precision. When we find such a network, we vary its number of neurons to observe how the precision of the ground-state energy changes, or stated differently: Is the number of neurons a suitable control-parameter to reliably increase or decrease the precision? This is interesting for potential simulations of larger system sizes, where we do not demand computational exact ground-state energies, to have an estimate for the precision we can expect for a certain number of parameters.

5.1. Test Outline

Neural networks approximate any square integrable function arbitrarily well with sufficiently many neurons, as we discuss in Sec. 4.2. This statement contains no information about what the actual relation between precision and number of neurons is. Further we also discuss in Sec. 4.2 that deeper networks can have an advantage over shallower networks, in terms of the parameter number. In many applications, like *computer vision*, it seems to be fruitful to build networks with as many layers as possible [42]. The limit of *depth* in these cases is just the increasing training complexity with the increasing number of layers, like we hint in Sec. 4.4.

To cover most scenarios of a reasonable neural network we test networks of different depth and with different activation functions. This requires a notation style for the different networks, Tab. 1 contains abbreviations that we use to refer to the associated activation function. Since the architecture can be defined by the number of layers and corresponding activation functions a four-layer architecture can be defined like

$$f^{(1)}f^{(2)}f^{(3)}f^{(4)} \; , \tag{5.2}$$

where the $f^{(i)}$ are then swapped for the actual abbreviations from Tab. 1. We use this notation style to label the architectures we use throughout this work. To investigate the effect of depth on a network it is not instructive to add a hidden layer that is activated with a different function than the other hidden layers. Thus, when we investigate depth we add only hidden layers with the same activation function. A five-layer network might therefore be denoted

$$f^{(1)}f^{(h)}f^{(h)}f^{(h)}f^{(2)} \; , \tag{5.3}$$

as this is quite longish we abbreviate the series of hidden activations with a power

$$f^{(1)}\left[f^{(h)}\right]^{\mathcal{L}-2}f^{(2)} \; . \tag{5.4}$$

In Eq. (5.4) we have of course $\mathcal{L} = 5$ for a five-layer network.

In Sec. 5.2 and Sec. 5.3 we investigate only networks of the form that we have in Eq. (5.4), and how their performance behaves when we increase the depth $\mathcal{L}$. To make the performance of networks comparable we need a way to calculate the layer sizes (number of neurons per layer) that depends on $\mathcal{L}$, because just adding layers of equal size allows deeper networks to have much more parameters than shallower networks. We resolve this issue by simply choosing a fixed number of total parameters $|\theta|$ and making all hidden layer equally large, this way we can calculate the layer size $n^{(1<i<\mathcal{L})} = M$ of the hidden layers by solving the equation

$$(\mathcal{L}-2)M^2 + (N + b(\mathcal{L}-1) + 1)M + b - \mathcal{D} = 0 \; . \tag{5.5}$$

where $\mathcal{D} = 2^N$ is the dimension of our Hilbert space and b a boolean variable, which tells whether we use bias or not. An example for the behavior of M for a given N can be seen in Fig. 5.1 and in App. C one can find a table that contains all layer-sizes that appear in our investigations. We choose $|\theta| = \mathcal{D}$ as an upper

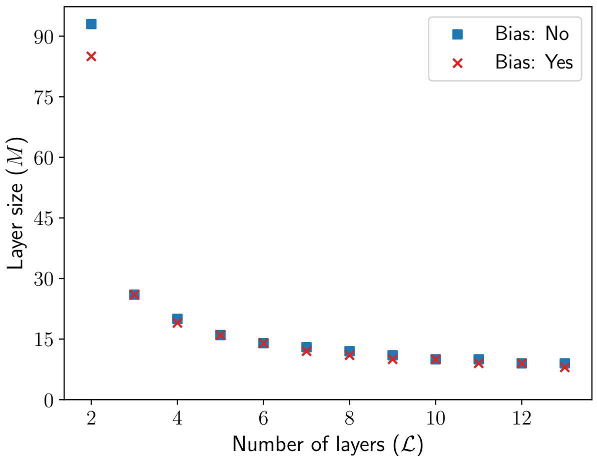


Figure 5.1: One can see here the layer-size M we get for a system size of $N = 10$.

bound for the number of parameters in the network. A network is actually not feasible for large system sizes when it requires $|\theta| \propto 2^N$ to work properly, but for now this is fine to tell whether a combination of activation functions works *at all*, we take care of reducing parameters at a later point.

This brings us to the two main types of tests we carry out with the networks we present in the course of this work:

1. We take a network with hidden layers of size M and make 20 (100 in some highlighted cases) attempts to find the ground-state energy. This way we get a histogram that tells us what the maximal possible precision for the network is and how probable it is to reach a certain precision.

2. As in 1. we repeat the optimization process 20 times, but here we do so for any layer configuration (and not just all hidden layers set to M), for which holds $|\theta| \leq \mathcal{D}$ (the total number of parameters being smaller than the Hilbert space dimension). We retain only the best obtained precision for every configuration.

The first test tells us whether a network is suitable to approximate the ground-state and the second test tells us whether the network might be feasible to tackle also larger system sizes. For both tests we initialize the network parameters with Gaussian distributed random numbers. Then the training is performed, with the hyper-parameters of Adagrad set to $\eta = 0.1$ and $\epsilon = 10^{-7}$ (for the explanation of these parameters see Sec. 4.3), for 10^6 gradient descent steps.

The expressiveness and training time of a neural network depends on the choice of the activation function. Thus, before we come to constructing and testing network architectures, we make some reasoning that allows us to argue why certain configurations are not tested. This reasoning concerns mostly the activation functions we employ and not so much the depth of the network. For this purpose Tab. 1 gives a quick overview over the most important properties of activation functions we use here and further on. Our discussion of the activation functions starts by considering the symmetry of a wave-function. Methods like Exact Diagonalization rely on exploiting symmetries of a Hamiltonian to reduce the complexity of finding its eigenstates [55, 41]. Also, in our case symmetries of the Hamiltonian translate to symmetries of the wave-function, which can be exploited to ease the approximation. But they can just as well be a constraint when one does not adapt the network to the given symmetries. For the components of the wave-function $q(\mathbf{v})$ of Hamiltonian $\mathcal{H}_{ms}$ holds the following

$$N = 4n + 2,\ n \in \mathbb{N}:\ q(-\mathbf{v}) = -q(\mathbf{v}) \tag{5.6}$$

$$N = 4n,\ n \in \mathbb{N}:\ q(-\mathbf{v}) = q(\mathbf{v})\ . \tag{5.7}$$

Equation Eq. (5.6) tells us that for system sizes of the form $N = 4n,\ n \in \mathbb{N}$ the wave-function is symmetric and for $N = 4n + 2,\ n \in \mathbb{N}$ we deal with an anti-symmetric wave-function. Now assume we have a neural network with a single layer and an arbitrary activation function

$$h_{W,0}(\mathbf{v}) = \mathbf{f}(\mathbf{W}\mathbf{v})\ . \tag{5.8}$$

When we now try to learn Eq. (5.6) with $h_{W,0}$ it is convenient to choose f to be an anti-symmetric function, since this ensures that the neural network has intrinsically the required symmetry. But should one try to approximate a case of the form Eq. (5.7), $h_{W,0}$ is limited due to its symmetry properties. Please note that, since we have not included any bias, the universality statement for neural networks does not hold and hence increasing the number of neurons does not lead to a better approximation. At this example one can already see how tedious the design of a suitable neural network can be, on one hand we can exploit the

Function	Symmetry	Output Range	Monotonic	Derivative Monotonic	$\propto x$ near zero
Sigmoid (S)	None	$(0, 1)$	Yes	No	Yes
Hyperbolic Tangent (T)	anti-symmetric	$(-1, 1)$	Yes	No	Yes
Rectified Linear Unit (R)	None	$[0, \infty)$	Yes	Yes	No
Cosines (C)	symmetric	$[-1, 1]$	No	No	No

Table 1

symmetry to get better results and on the other we lose the expressiveness for the opposed symmetry case.

But to get started we apply the transformation from Sec. 2.2 to Hamiltonian $\mathcal{H}_{ms}$ to get $\mathcal{H}_{ps}$ which yields a positive q. That way Eq. (5.7) holds for all even N and we can make some reasoning about which activation can be used in which way, without taking symmetry properties too early into account. So we start in Sec. 5.2 with only considering $\mathcal{H}_{ps}$ and then move on to also consider $\mathcal{H}_{ms}$ in the next sections.

5.2. Sign Positive Wave-Function

In the introduction to this section we state that the ground-state wave-function of Hamiltonian $\mathcal{H}_{ps}$ has only either positive or zero components. To emphasize this one can see the exact wave-function for $N = 8$ in Fig. 5.2. In order to make a two-dimensional plot of the wave-function, we enumerate the state configurations. The enumeration plays no particular role for the network, the only order we respect is that when a state $\mathbf{v}$ has the number i, the state $-\mathbf{v}$ has the number

$$(i + \mathcal{D}/2) \mod \mathcal{D} \,, \tag{5.9}$$

this way we account for the symmetric character of the wave-function. Conveniently enumerating configurations by translating their binary value into the decimal system respects the symmetry thus we stick to this system. We begin with the architecture that is composed only of T activation functions. An example for a three-layer network of this kind can be seen in Fig. 5.3. This network is tested

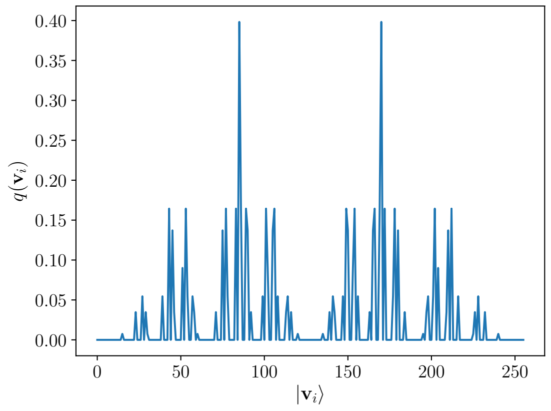


Figure 5.2: The figure shows the amplitude q for the ground-state of the Hamiltonian $\mathcal{H}_{ps}$ for $N = 8$.

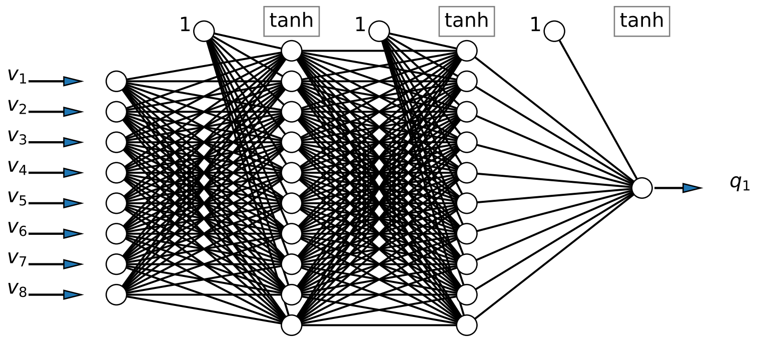
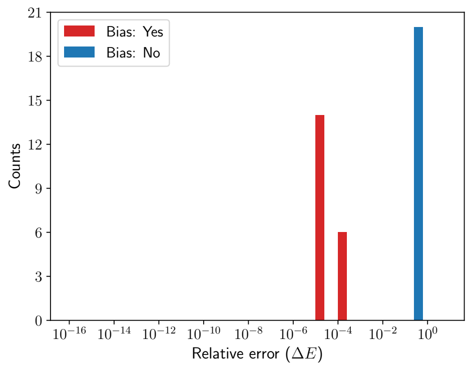
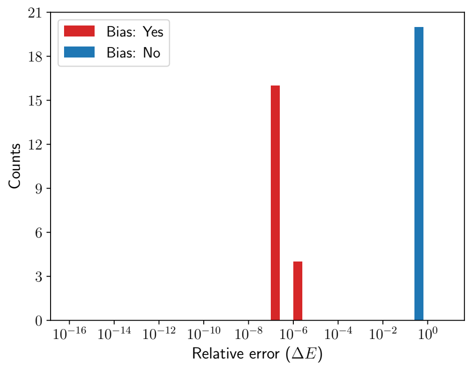


Figure 5.3: The picture shows an example for a three-layer network, composed only of tanh activation functions. In terms of the abbreviated notation this architecture reads TTT.

with three layers, as in Fig. 5.3, and in a two, four and five-layer configuration. The results of these tests are shown in Fig. 5.4-Fig. 5.5.



(a) Model: TT

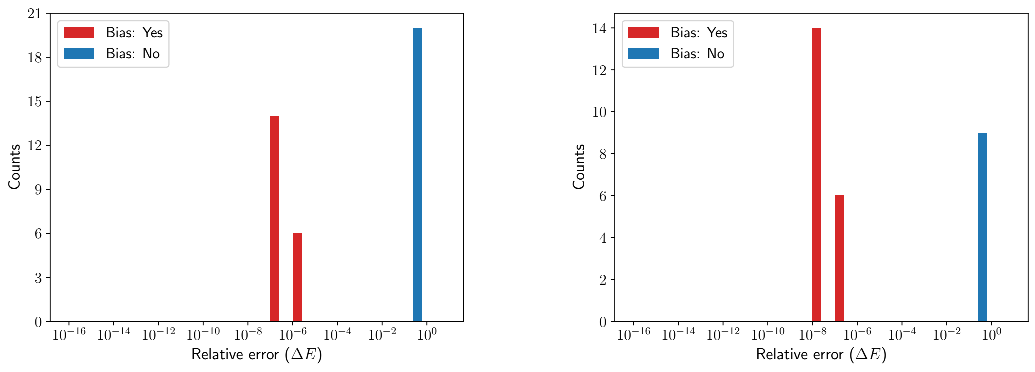


(b) Model: TTT

Figure 5.4: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$

The first observation we make in Fig. 5.4 is that a network of the form Fig. 5.3 is strongly bounded in its capability to approximate a symmetric function without the use of bias. We try to emphasize this instance with intuitive arguments (Eq. (5.8)) and in Fig. 5.4 one can see the actual impact of the intrinsic symmetry of an activation function on the approximation capabilities of a network. But when bias is enabled, the symmetry restrictions can be overcome. In section Sec. 4.2 we give a short hand-waving explanation why deeper networks have sometimes advantages over shallower networks. In Fig. 5.4b we use a network that has the same number of parameters, but one more layer than the network used in Fig. 5.4.

In this case more depth seems to be an advantage. Adding a layer does actually improve the best reachable precision from $\Delta E = 10^{-5}$ to $\Delta E = 10^{-7}$, hence an improvement of magnitude 2.



(a) Model: $\mathrm{T(T)^2T}$ **(b)** Model: $\mathrm{T(T)^3T}$

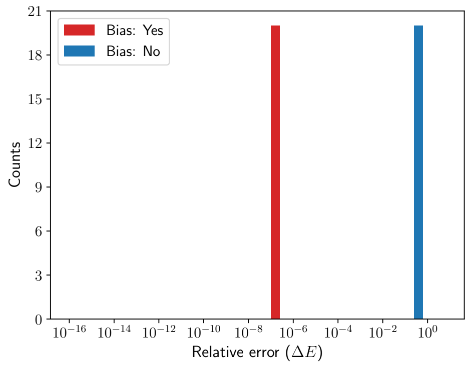
Figure 5.5: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$

By further increasing the network depth by one layer, like we do in figure Fig. 5.5, we get no improvement in precision but loss of reliability. This can be seen by comparing the height of the bars corresponding to the precision $\Delta E = 10^{-7}$ in Fig. 5.4b and Fig. 5.5a. In other words, the best precision, that is reached, becomes less probable to occur, which is of course something we do not favor. When we then go to $\mathcal{L} = 5$ (Fig. 5.5b) we gain one magnitude in precision, compared to Fig. 5.5a. But this additional magnitude comes for the cost of a lower reliability than in Fig. 5.4b. By further increasing the network depth we do not gain any more precision, but solely loose reliability (this can be seen in Fig. 5.12).

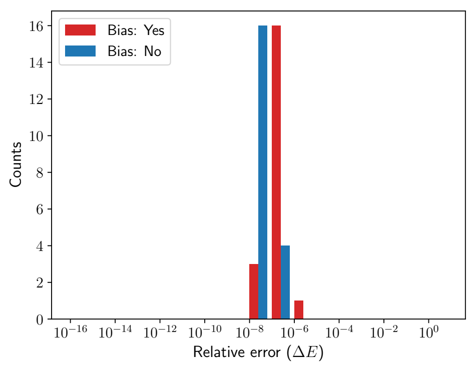
Now we repeat the same process but instead of T activation functions we use only S activation functions. The results can be seen in Fig. 5.6. The S activation function has no intrinsic symmetry (Tab. 1), thus we do not expect that the network performs with bias better than without. We expect that the models (with bias respectively without bias) perform equally and regarding the precision we expect similar performance as we see with the T based network. This is due to the instance that, by employing the bias, we can express T with S

$$\mathrm{T}(x) = 1 - 2\mathrm{S}(-2x) \ , \tag{5.10}$$

and also the other way around, thus we can assume a similar performance as with T activation functions.



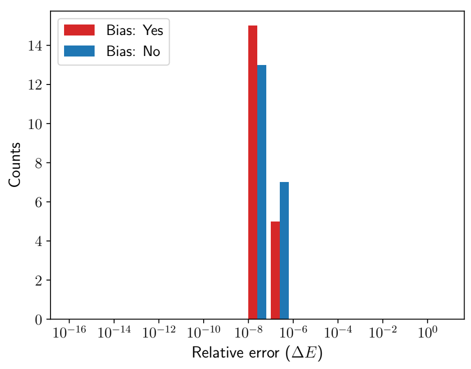
(a) Model: SS



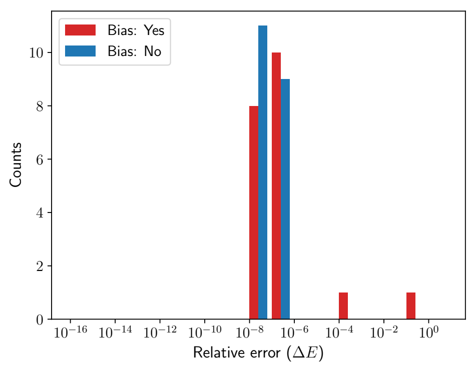
(b) Model: SSS

Figure 5.6: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$

In Fig. 5.6 it turns out that we are right with our estimation of the precision, we do actually get the same maximal precision in Fig. 5.6b as in Fig. 5.5. But it also turns out that for a network with two layers and no bias we are bounded to a lower precision, as Fig. 5.6a. This implies that bias is not only useful to break symmetries but also to create symmetries, like the spin-inversion symmetry in this case. When we move to $\mathcal{L} = 3$ (Fig. 5.6b) our guess about the bias seems to be right. It happens to be such that the network without bias is even more reliable in yielding the best possible precision, than the network without. At this point the impact of depth is observable. The S network is with three layers much more powerful than with two, even though they consist of the same number of parameters.



(a) Model: S(S)^{2}S



(b) Model: S(S)^{3}S

Figure 5.7: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$

So when we have only T activation functions we have a gain of one magnitude in precision when going form $\mathcal{L} = 4$ to $\mathcal{L} = 5$. The thereby highest found precision is $\Delta E = 10^{-8}$ and we expect the S based network not to exceed this precision, especially at this depth there is no qualitative difference between S and T. Fig. 5.7 proves our assumptions right. Thus, we conclude from this results that deeper networks can overcome limitations that a network with just one layer less is bounded to. Increasing depth does also make the network more general as the intrinsic characteristics of the activation function do not matter that much anymore (compare Fig. 5.5b and Fig. 5.7). Though we find a sweet-spot for depth until which precision and reliability increases and after which precision does not increase anymore and reliability degrades.

Until this point the symmetry of an activation function is the only feature of the function we worry about. Another important feature of a function is its derivative. Since the employed optimization algorithm relies on the calculation of the gradient of the activation functions, it is an important point that the gradient should not die out. Functions like S and T have a vanishing derivative for arguments with high absolute values. The R activation has a constant non-zero gradient for arguments with positive sign. This property makes it very convenient to build into a neural network (see Sec. 4.4 for greater detail). But a drawback is the unbounded character of R, as the wave-function components can only take values in $[0, 1]$ (for the particular case of the ground-state of Hamiltonian $\mathcal{H}_{ps}$) it is not reasonable to expand the search-space to $[0, \infty[$. But an even less desired tendency of R is to map values to exact zeros. As all negative values are mapped to zero it is very likely that half of the configurations from $\mathcal{T}$ are mapped to zero, when R is the activation of the first layer. For large enough biases $\mathbf{b}^{(1)}$ this is not the case, but then R acts as a regular linear function and does not exhibit any non-linearity. For these two reasons we do not use R as activation function in the first or last layer,

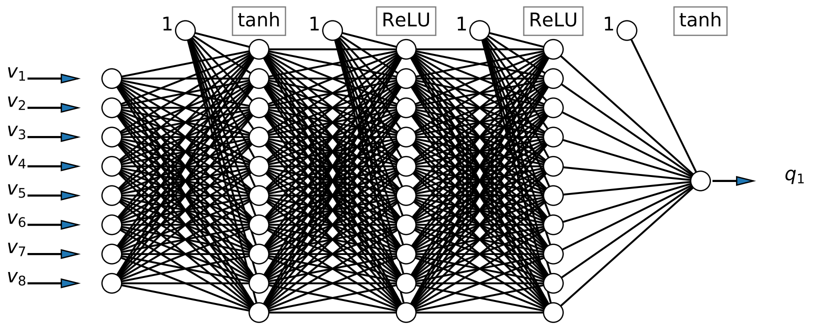
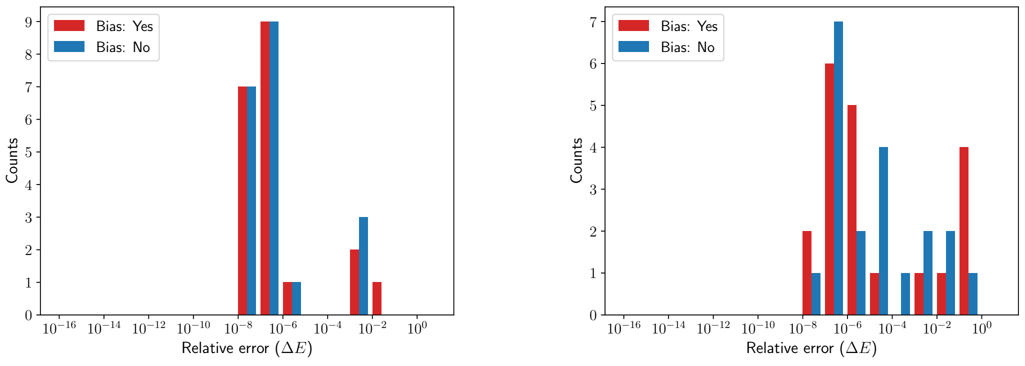


Figure 5.8: The picture shows an example for a four-layer network, composed only of R and T activation functions. In terms of the abbreviated notation for architectures, this network reads TRRT.

but we sandwich it between activation functions that are bounded and always non-zero for non-infinite or non-zero values respectively. In Fig. 5.8 we provide an example of a four-layer network for such a sandwich architecture.

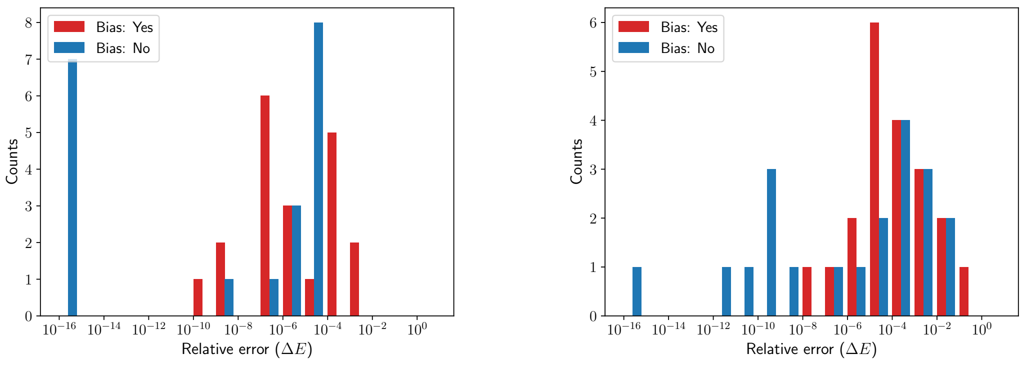
First, we discuss the case when R is sandwiched between S functions. The results for these tests can be seen in Fig. 5.9.



(a) Model: SRS **(b)** Model: S(R)^{2}S

Figure 5.9: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$

In Fig. 5.9a one can see that we do not obtain higher precision than in 5.6b, but we obtain the same maximal precision with higher probability. When we increase the network depth this is not true anymore, as Fig. 5.9b shows. The other possibility we have for a sandwiched network is the combination of R and T.

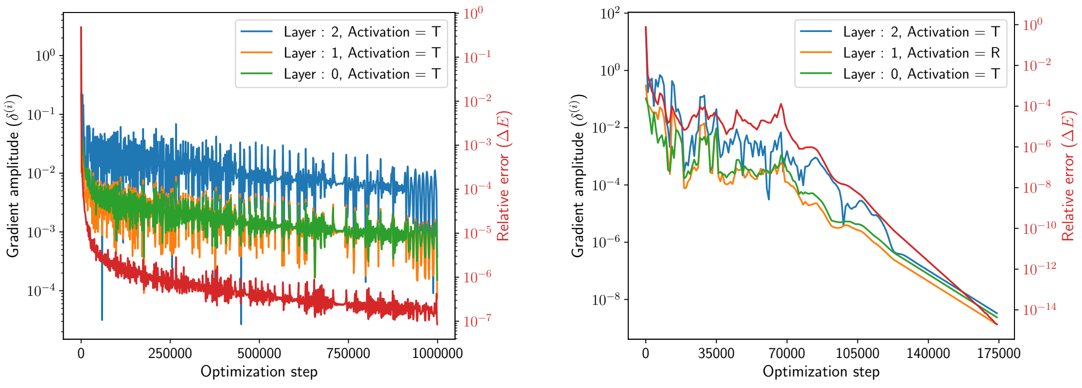


(a) Model: TRT **(b)** Model: T(R)^{2}T

Figure 5.10: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$

This combination allows us to fully exploit the great properties of R. In Fig. 5.10a one can see that this architecture yields the best precision so far, which is in fact *machine precision* ($\Delta E = 10^{-16}$). It does so while the precision that appears with almost the highest probability being the highest precision. These are some very favorable properties of a network but cannot be improved (actually the opposite occurs) when adding one layer as we see in Fig. 5.8.

We mention quite some time that the R offers an advantage in the calculation of the gradient of the neural network. Besides the explanation in Sec. 4.4 we show in Fig. 5.11 that the advantage of R can be made visible, and not just by observing higher precision. The networks TTT and TRT are very similar, besides that the best precision that TTT yields is $\Delta E \approx 10^{-7}$. But the similarity to TRT is striking and we get, by adding only one R activated layer, a precision of about $\Delta E \approx 10^{-16}$, so it is fair to ask why? The answer can be obtained by observing how the absolute values of the $\delta^{(i)}$ evolve during the optimization. This is what we do in Fig. 5.11.



(a) Model: TTT **(b)** Model: TRT

Figure 5.11: In this figure we see the norm of the gradients that are calculated in every optimization step and for every layer. At the same time one, can see in red the evolution of the relative error. The treated Hamiltonian for this case is $\mathcal{H}_{ps}$ with $N = 8$.

What we see in Fig. 5.11a is a combination of the weak gradient of T with the way we calculate the learning rate Eq. (4.36). Due to the convergence of T towards 1 for large values, the gradient dies out. When this process starts the update Eq. (4.36) does not degrade the learning rate too strongly, but this is not enough to cope with the small gradient we obtain. The result is that the optimization process stops at a certain precision, due to the strong monotonic decay of Adagrad. Another typical symptom of the vanishing gradient problem is that the gradient of the last layer is significantly larger than the gradient of all other layers (see [53, 4] for more information about the vanishing gradient problem). The behavior in Fig. 5.11b is in that sense very different. Due to the strong gradient of R, we find a strong gradient at the beginning and a vanishing one once we approach the minimum.

Also, one can see that the norm of the gradients of all layers are about the same, in contrast to the case of the TTT network.

Let us summarize this section with Fig. 5.12. In this figure one can see the best precision (smallest error relative to the exact ground-state energy) every network, we test in this section, is able to yield no matter with or without bias. Further one can see the probability with which this precision can be obtained. These results are plotted against the layer-size, architectures that consist of two different activation functions start at three layers. We indicate the amount hidden layers by raising the hidden activation function to the power of $\mathcal{L} - 2$.

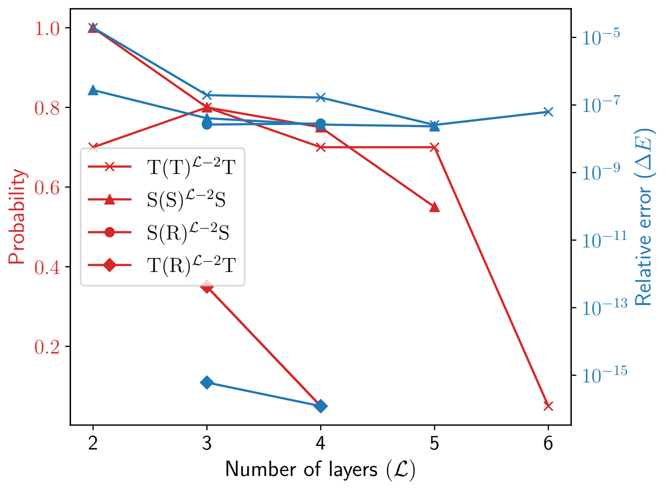
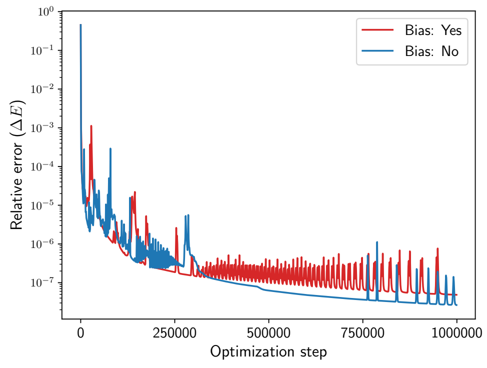
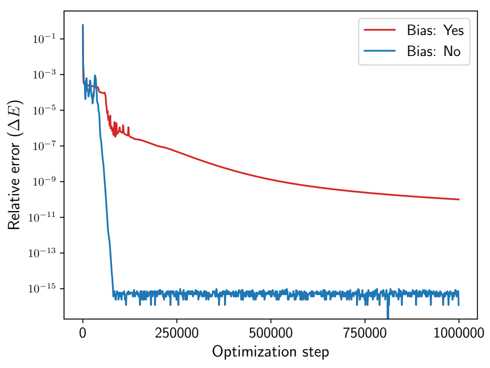


Figure 5.12: In this figure one can see the precision that the here tested networks yield at best (with and without bias combined) for increasing depth on the x-axis. In red one can see the probability with which the corresponding probability is obtained.

Fig. 5.5b shows two things quite clearly, first, depth improves precision until a certain point, secondly, depth makes the training process more cumbersome. The second point is deduced from the fact that the probability for a certain precision decreases with increasing depth. In Fig. 5.5b the similarity between homogeneous T and S networks is emphasized again. Their striking similarity is contrasted by the performance difference they show in combination with R. One possibility for this difference might root form the SRS optimization not being converged, but Fig. 5.13 shows the relative error of TRT and SRS during the optimization process and both seem to have converged. Thus, we simply have to accept that R performs better in combination with T than with S. We conclude this discussion with the insight that TRT is a good combination of depth and powerful gradient, which makes it the best network so far.



(a) Model: SRS

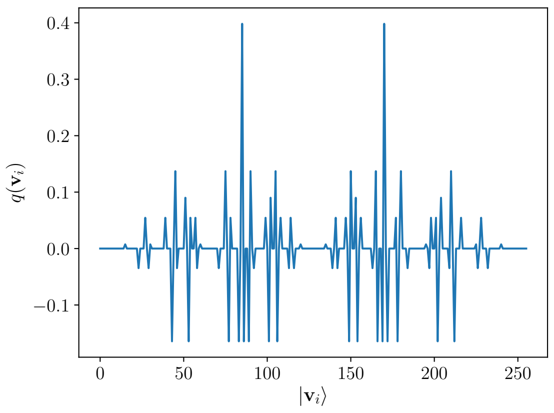


(b) Model: TRT

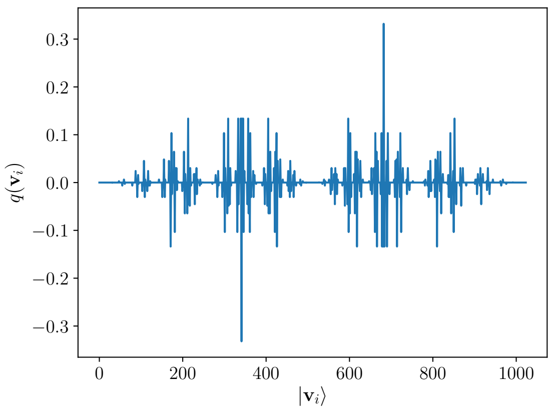
Figure 5.13: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$; In this figure one can see the relative ground-state energy error during the optimization of a SRS (Fig. 5.13a) and a TRT network (Fig. 5.13b).

5.3. Mixed Sign Wave-Function

Now that we have gained some insights on the behavior of our activation functions and the impact of depth, we can take on the next difficulty, the alternating sing case. We return to Hamiltonian $\mathcal{H}_{ms}$ which has a ground-state wave-function that involves also negative values and follows the rules from Eq. (5.6) respectively Eq. (5.7). How the actual wave-functions look like can be seen in figure Fig. 5.14, where Fig. 5.14a corresponds to a case of Eq. (5.7) and Fig. 5.14b to Eq. (5.6). Thus



(a) System size: $N = 8$



(b) System size: $N = 10$

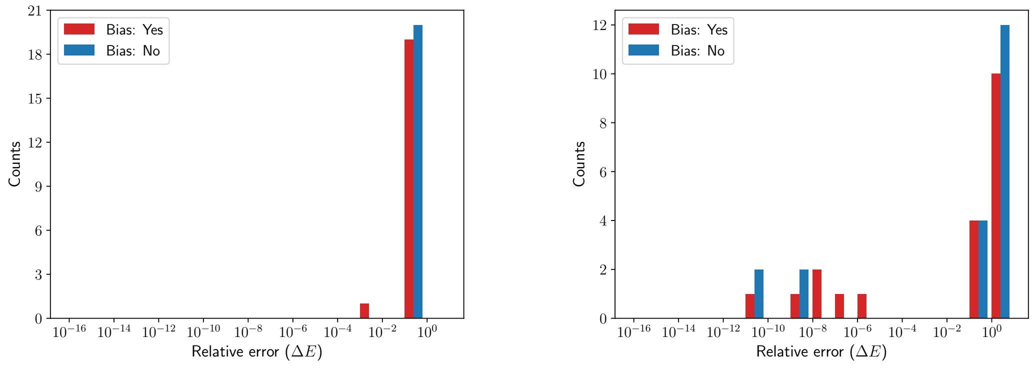
Figure 5.14: Here one can see the wave-function components q_i for the ground-state of Hamiltonian 2.17, for a system size of $N = 8$ and $N = 10$.

Function / Usage	Hyperbolic Tangent	Sigmoid	ReLU	Cosinus
First Layer	x	x		x
Hidden Layer	x	x	x	x
Output Layer	x			x

Table 2

we need to construct a network that can output negative values and is unbiased towards symmetry. The first condition can be met by having always a T as output activation function (or a C activation, but this is subject of later discussion). From that and the already established restrictions we get the rules in Tab. 2 for the composition of a neural network. Now we build networks according to Tab. 2 and test them as before in Sec. 5.3. This time we test not just $N = 8$, but also $N = 10$ system sizes to cover both symmetry cases.

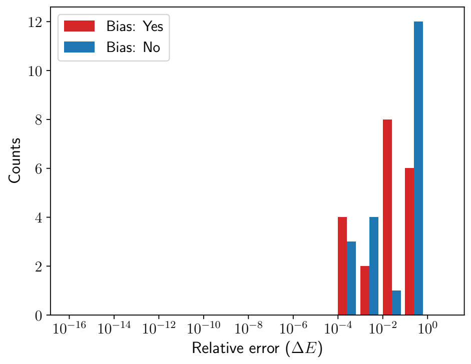
We start the discussion with a network composed of S functions (besides the output of course).



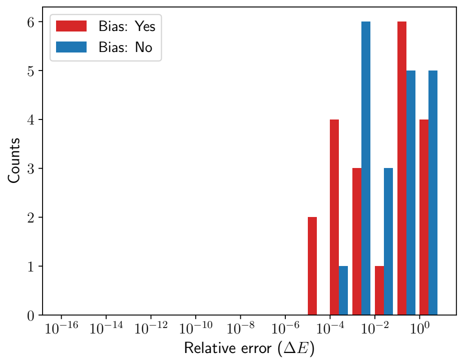
(a) System size: $N = 8$ **(b)** System size: $N = 10$

Figure 5.15: Model: ST, Hamiltonian: $\mathcal{H}_{ms}$

The S function has no symmetry, thus there is no symmetry that needs to be broken by the bias or other layer. Though Fig. 5.15 reveals that the simple two-layer architecture has a preference for anti-symmetric functions Fig. 5.15b.



(a) System size: $N = 8$

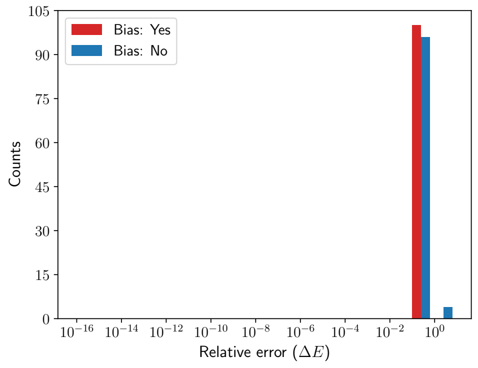


(b) System size: $N = 10$

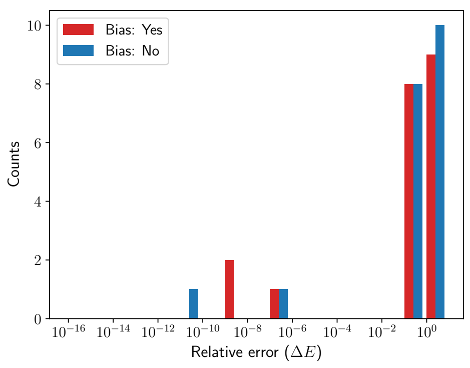
Figure 5.16: Model: SST, Hamiltonian: $\mathcal{H}_{ms}$

Adding one layer leads to a better performance on the symmetric case Fig. 5.16b. But as Fig. 5.16 shows we achieve this only by trading off performance on the anti-symmetric case Fig. 5.16a. Thus, we stop here with this combination of activation functions and proceed with a different activation.

The T activation is an anti-symmetric function, Fig. 5.17 shows this property quite clearly. The tests on the-anti symmetric case Fig. 5.17b yield some good results, while the tests on the symmetric case Fig. 5.17a show complete failure. Interestingly TT preforms not as bad on the symmetric wave-function with positive components (Fig. 5.4a), thus it struggles due to the negative values.



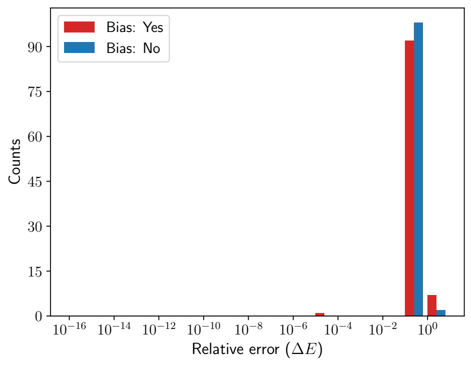
(a) System size: $N = 8$, Number of optimization attempts: 100



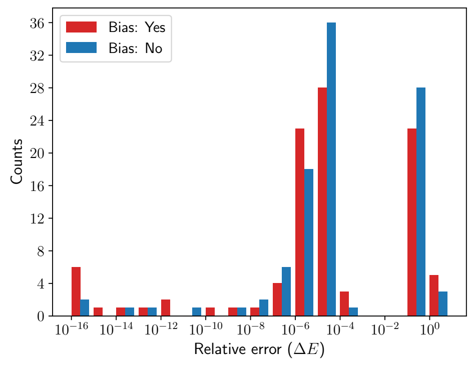
(b) System size: $N = 10$, Number of optimization attempts: 20

Figure 5.17: Model: TT, Hamiltonian: $\mathcal{H}_{ms}$

When we add another T layer to the network, we obtain a three-layer network as in Fig. 5.3. This network delivers way better performance on the preferred symmetry case Fig. 5.18b. While it still does not great on the symmetric case, we see in Fig. 5.18a that with a very small probability it yields a non-trivial precision.



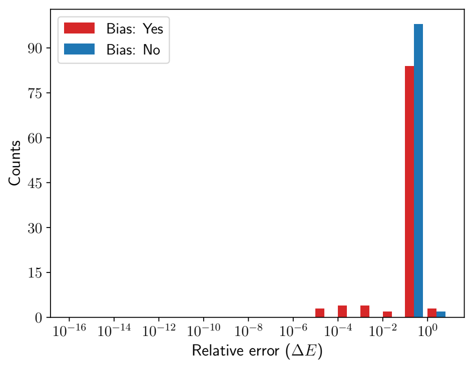
(a) System size: $N = 8$



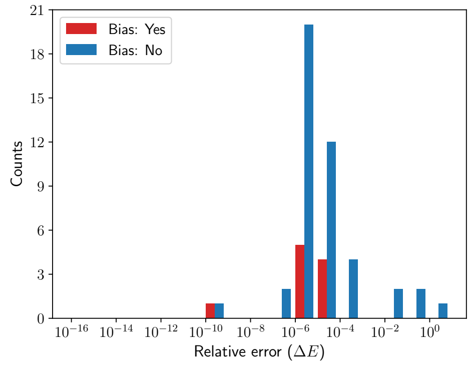
(b) System size: $N = 10$

Figure 5.18: Model: TTT, Hamiltonian: $\mathcal{H}_{ms}$, Number of optimization attempts: 100

Fig. 5.19a is good example for the impact of depth. Even if we do not change the composing functions we can overcome the symmetry restraints of the functions to represent functions with different symmetry. This is only possible due to the application of biases, as one can see from Fig. 5.19a, which allows us to break the symmetry. For four T layers it seems like we have not lost any of the approximation capabilities on the anti-symmetric case Fig. 5.19b.



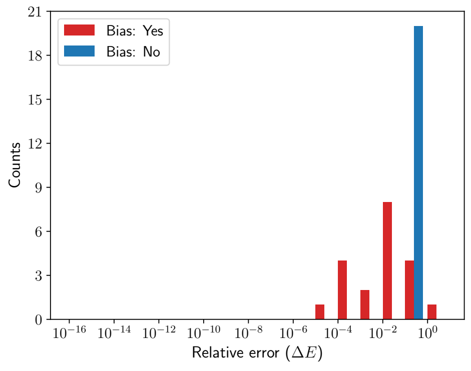
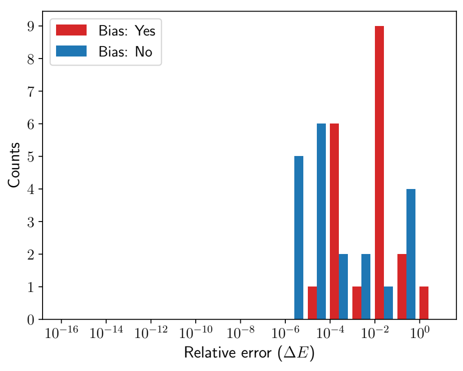
(a) System size: $N = 8$



(b) System size: $N = 10$

Figure 5.19: Model: $\mathrm{T(T)^2T}$, Hamiltonian: $\mathcal{H}_{ms}$, Number of optimization attempts: 100

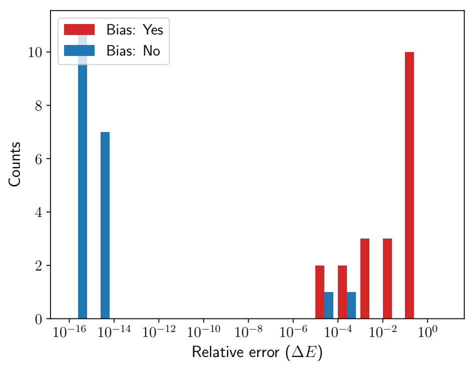
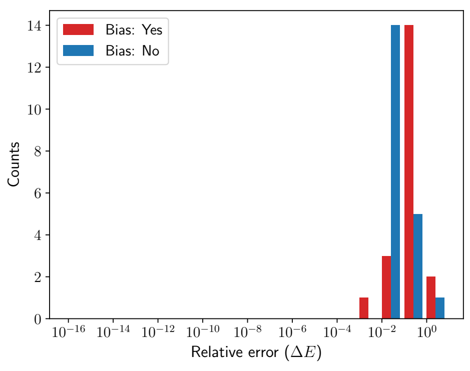
From Fig. 5.20 we learn that four layers is the sweet spot for this activation function, since adding more layers does not improve the results. As we test also the eight and twelve layer configuration (Fig. 5.20, Fig. 5.23) we observe that the degrading continues, thus for our training procedure, stacking four T layers is the best one can do.



(a) System size: $N = 8$ **(b)** System size: $N = 10$

Figure 5.20: Model: T(T)^{4}T, Hamiltonian: $\mathcal{H}_{ms}$

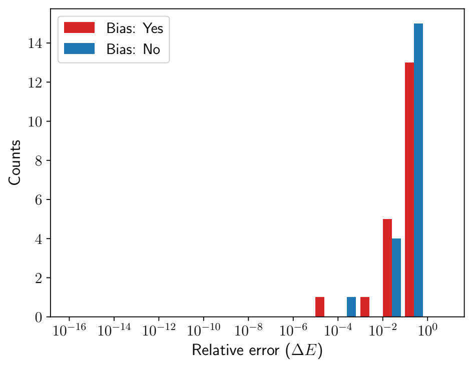
Another possible configuration is sandwiching several R activation functions between T activation (see Fig. 5.8 for an example). For a single hidden layer Fig. 5.21 we obtain the clearest histogram of all our trials. Fig. 5.21a shows that R allows to break the preference of T for the anti-symmetric functions and is also very favorable in terms of training, in correspondence to what we see in Sec. 5.2. Though the performance on the anti-symmetric case Fig. 5.21b is rather poor.



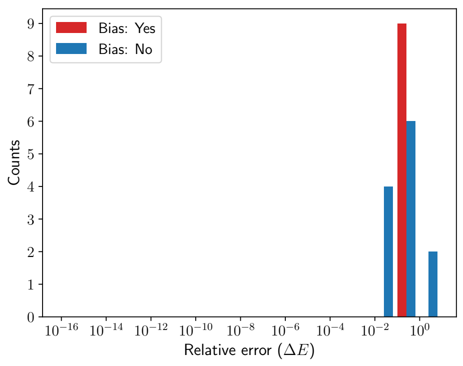
(a) System size: $N = 8$ **(b)** System size: $N = 10$

Figure 5.21: Model: TRT, Hamiltonian: $\mathcal{H}_{ms}$

When testing two R layers in Fig. 5.22 we were do not only fail to improve on the anti-symmetric case but degrade also performance on the symmetric case. Thus, it seems like the three-layer architecture is still the sweet spot for this configuration.



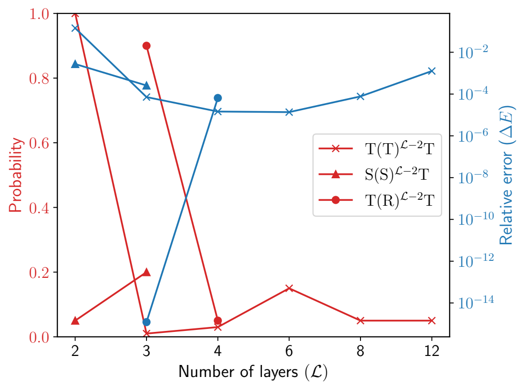
(a) System size: $N = 8$



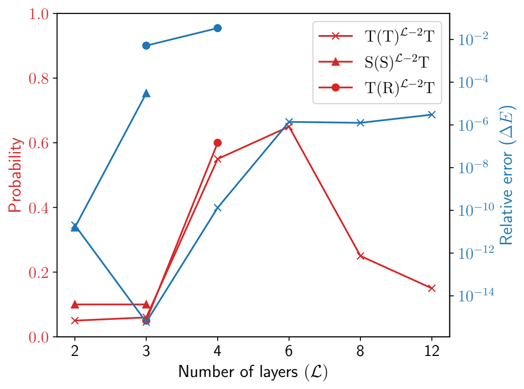
(b) System size: $N = 10$

Figure 5.22: Model: T(R)^{2}T, Hamiltonian: $\mathcal{H}_{ms}$

We sum up all the results of this section in Fig. 5.23. The hereby presented figures follow the same concept of Fig. 5.12, we show how reliability and precision behave with increasing network depth.



(a)



(b)

Figure 5.23: In this figure one can see the precision that the here tested networks yield at best (with and without bias combined) for increasing depth on the x-axis. In red one can see the probability with which the corresponding probability is obtained. Fig. 5.23a shows the results for $N = 8$ (symmetric case) and Fig. 5.23b does the same for $N = 10$ (anti-symmetric case).

In correspondence to Fig. 5.12 both figures in Fig. 5.23 show that, for some architectures, increasing depth improves the obtained precision. In the case of TTT in Fig. 5.23b this is shown impressively. But it becomes also evident that there is a sweet-spot for depth, after which precision degrades again. For the ST network it happens to be so that increased depth does not lead to better performance. The ST network stands out, due to its two-layer architecture. It is also interesting, since the S has no preferred symmetry and though it performs way better on the anti-symmetric than on the symmetric problem. In the next section we try to optimize the architecture of two-layer networks as far as possible, as they are favorable in terms of training. The other network that stands out in all tests so far is the TRT network. An optimal network is one that performs on both symmetry cases like TRT does on the symmetric case. This is what we try to find in the next part of this sections.

5.4. Exploiting Symmetries and Functions with Strong Gradient

In the previous part of this section we learn which activations can be combined and which depth is favorable regarding certain activation configurations. In this section we use this knowledge to construct architectures we think might work well and then test their reliability and how many parameters they need to accomplish the approximation. The latter test is number 2 in the list in Sec. 5.1, recall that as before we repeat the optimization process 20 times per network but this time we do so for any layer configuration, for which holds $|\theta| \leq \mathcal{D}$ (the total number of parameters being smaller than the Hilbert space dimension). We display the best obtained precision for every configuration encoded with a color scheme.

5.4.1. Tanh - Cos

In Fig. 5.15 we have a two-layer network, that outperforms all two-layer networks that we test so far. It performs particularly well on the anti-symmetric case $N = 10$. The ST network has a layer with no symmetries as input and a layer with anti-symmetry as output. Now we would like to build a network out of two layers and it shall have as input activation an anti-symmetric function and as output a function that can be either symmetric or anti-symmetric. The beneficial thing about deep architectures is, as we were able to see in 5.19a, that the characteristics of the activation function do not matter anymore at a certain depth. This makes Deep Neural Networks a suitable ansatz to infer highly complex functions we have no knowledge about and are willing to settle for decent precision. But here we do have some knowledge about the ground-state wave-function of Hamiltonian $\mathcal{H}_{ms}$ and are not willing to settle for decent precision, but demand machine precision. First of all, we want to use the symmetry aspect of the ground-state wave-function. This can easily be exploited by having the following architecture

$$h_{W,b}(\mathbf{v}) = \cos(\mathbf{w}^{(2)} \cdot \tanh(\mathbf{W}^{(1)}\mathbf{v}) + b^{(2)}). \tag{5.11}$$

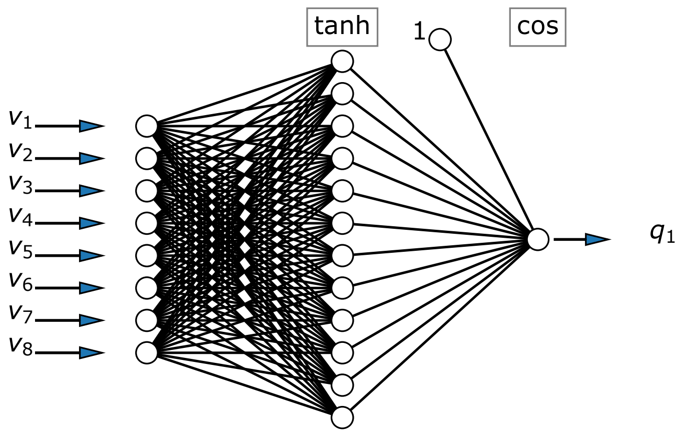
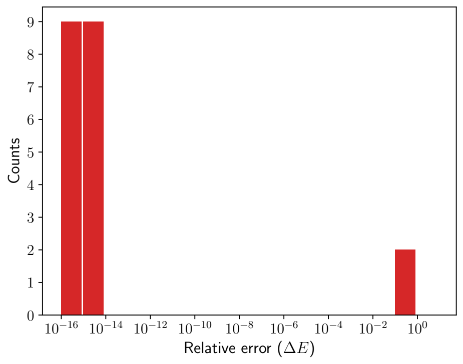
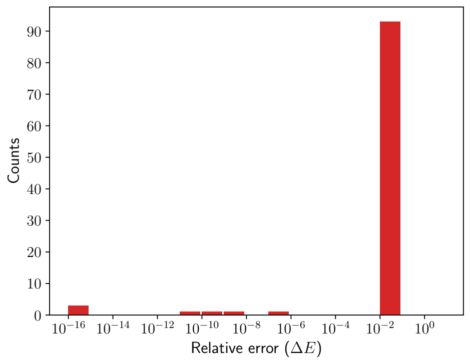


Figure 5.24: The picture shows an example for a two-layer network with a TC architecture.

Fig. 5.24 shows what network 5.11 looks like in the usual representation. The idea behind this is that the T retains the anti-symmetric character of the input, while the C transforms the output in a symmetric function for $b^{(2)} = 0$ and an anti-symmetric function for $|b^{(2)}| = \pi/2$. Through setting the bias by hand one can force the network to either learn a symmetric or an anti-symmetric function. A drawback of this is of course the loss of generality regarding wave-functions with no such symmetry. Anyway, it is favorable having a *shallow* (shallow is the opposite of deep) network, like this one. Shallow networks do not suffer from the vanishing gradient problem (see Sec. 4.4) like deeper networks do. Thus, training is not so cumbersome. Network Eq. (5.11) has only one hidden layer, and the size of this layer is the only that can be varied. First we set this layer to $n^{(1)} = M = (\mathcal{D} - 1)/(N + 1)$ to find out which precision can be reached and with which probability. In figure Fig. 5.25 we test on Hamiltonian $\mathcal{H}_{ms}$ with system sizes $N = 8$ and $N = 10$.



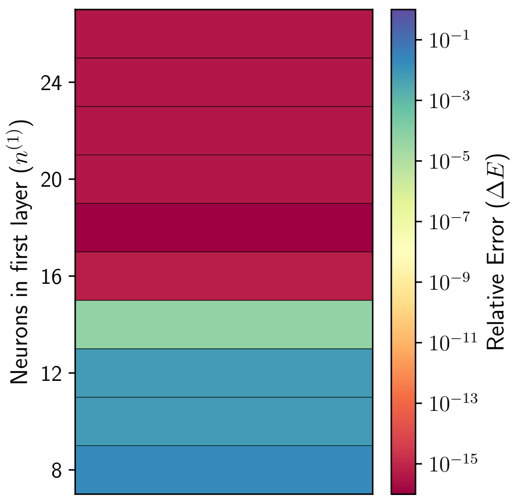
(a) System size: $N = 8$, Number of optimization attempts: 20



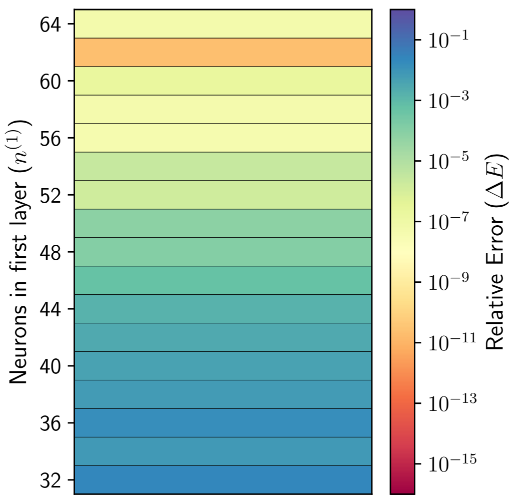
(b) System size: $N = 10$, Number of optimization attempts: 100

Figure 5.25: Model: TC, Hamiltonian: $\mathcal{H}_{ms}$

The first thing one notices in Fig. 5.25 is that for $N = 8$ we get similar reliability with TC as in Fig. 5.21a for TRT, which is a great result, but does not translate to the anti-symmetric case of $N = 10$. Other than in all previous cases the TC network can reach machine precision on both symmetries, even if it is not very reliable on the anti-symmetric case. Thus we proceed now to find out how small $n^{(1)}$ can be, such that we still obtain machine precision.



(a) System size: $N = 8$



(b) System size: $N = 10$

Figure 5.26: Model: TC, Hamiltonian: $\mathcal{H}_{ms}$; In this figure the color refers to the smallest relative error one obtains when initializing and optimizing a network 20 times.

The results of this test can be seen in Fig. 5.26. For the symmetric case Fig. 5.26a net network yields machine precision with $n^{(1)} = 16$ hidden nodes, which translates to a total of $|\theta| = 145$ parameters. In the anti-symmetric case Fig. 5.26b we have a successive improvement with growing layer-size, but we never reach machine precision. This is due to the difficulty of estimating the right bias. If we are able to fix this problem, we have the first network that can deal with both symmetry cases equally. This is what we aim for in the next section.

5.4.2. Tanh - Triangle

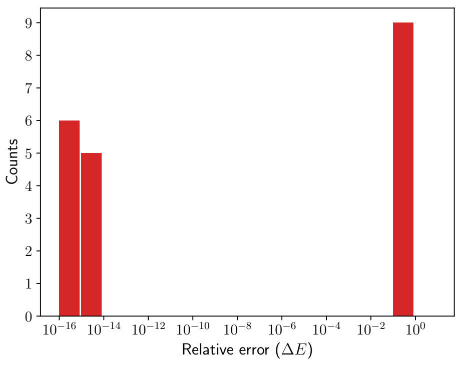
In 5.21a one can see impressively what advantageous can arise from a function that has a not-vanishing gradient. It might not improve the results of the approximator but raise the probability of reaching the best possible precision. The TC network, presented in the previous section, is a good approximator but it is cumbersome to train, due to vanishing gradients. Thus, it is straightforward to search for a function with the same properties as the C, but with a stronger gradient. The answer to this is easily found to be the *triangle-signal function*

$$D : x \mapsto 2|x \mod (2\,\mathrm{sign}(x)) - \mathrm{sign}(x)| - 1 \,, \tag{5.12}$$

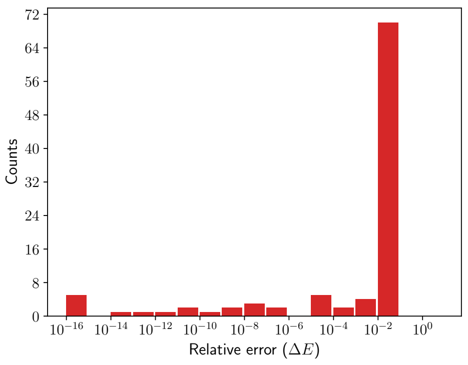
with the convenient gradient

$$\frac{\mathrm{d}D}{\mathrm{d}x} : x \mapsto 2\,\mathrm{sign}(x \mod (2\,\mathrm{sign}(x)) - \mathrm{sign}(x)) \,. \tag{5.13}$$

The triangle-signal has the same symmetry properties as the cosines, but its gradient is either -1 or 1, thus it is way less likely that an optimization process gets stuck. To check whether we get an improvement we repeat the test of Fig. 5.25 with the TD architecture.



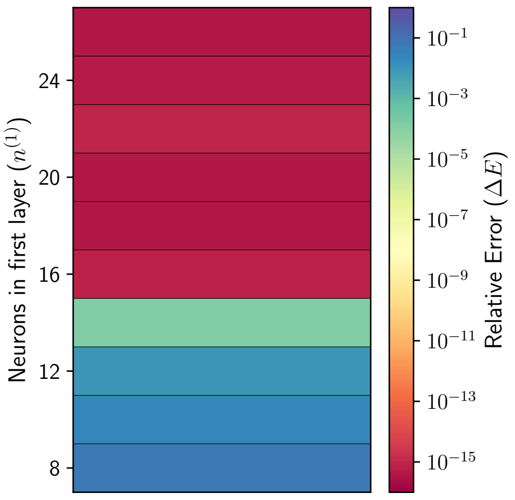
(a) System size: $N = 8$, Number of optimization attempts: 20



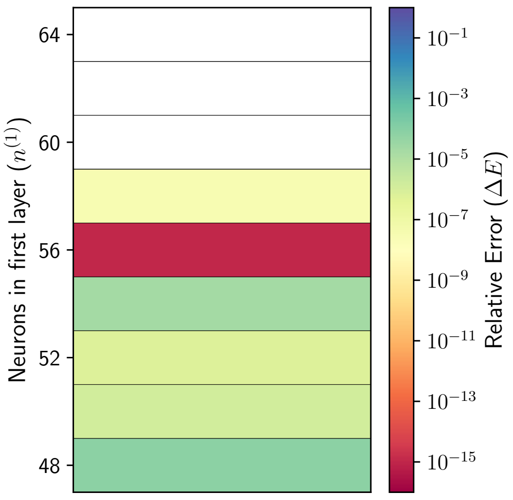
(b) System size: $N = 10$, Number of optimization attempts: 100

Figure 5.27: Model: TD, Hamiltonian: $\mathcal{H}_{ms}$

For the symmetric case we find in Fig. 5.27a the same precision and nearly the same reliability as in Fig. 5.27a. More importantly Fig. 5.27b shows that the D does improve the reliability to get to machine precision, compared to Fig. 5.25b. We get also more frequently better precision than $\Delta E \approx 10^{-1}$, which is the precision TC surpasses only hardly.



(a) System size: $N = 8$



(b) System size: $N = 10$

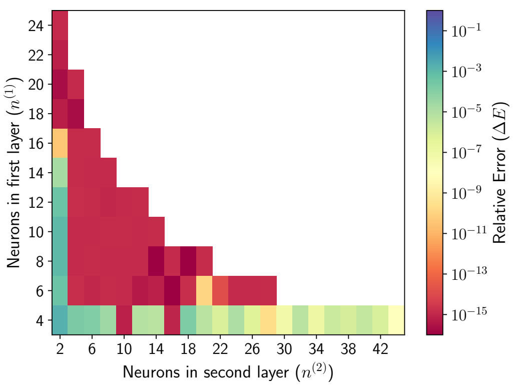
Figure 5.28: Model: TD, Hamiltonian: $\mathcal{H}_{ms}$; In this figure the color refers to the smallest relative error one obtains when initializing and optimizing a network 20 times.

Finally, we want to know how small we can make the first layer, such that we still reach machine precision. Fig. 5.28a shows that for $N = 8$ the smallest required number neurons is $n^{(1)} = 16$, which is the same as for the TC network. Fig. 5.28b shows how difficult the anti-symmetric case is, since once we reach machine precision for $n^{(1)} = 56$ neurons we do not reach it for the next layer-sizes, even though we initialize 20 times for every such data point. Thus, we need to further improve on the gradient or the way the symmetry is determined.

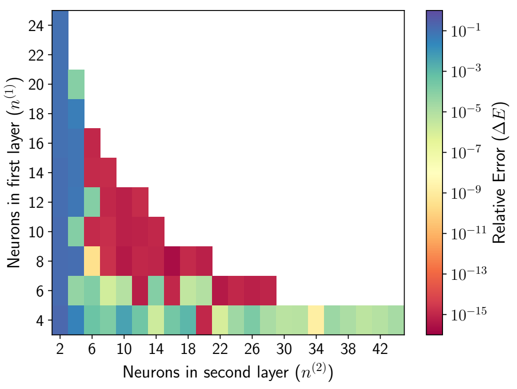
5.5. Exploiting Redundancy and Depth

5.5.1. Tanh - ReLU - Tanh

In this subsection we review the architecture form Fig. 5.10a. While this architecture was not the most versatile, it is the most reliable to find a representation of q with machine precision. In Sec. 5.4.2 we explain a way to tackle the symmetry problem, but do not find a satisfying solution for the reliability problem. Thus, we investigate the TRT architecture, because it is very reliable on the symmetric case while it does not exploit any symmetry. We start our investigation by searching for the smallest total number of parameters that still allows TRT to yield machine precision. While in Fig. 5.10a and Fig. 5.21a the number of parameters we use to accomplish the approximation corresponds to the Hilbert space dimension, in Fig. 5.29 one can see that in fact the same precision can be reached with way less parameters.



(a) Model: TRT, Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$



(b) Model: TRT, System size: $N = 10$

Figure 5.29: Hamiltonian: $\mathcal{H}_{ms}$; In this figure the color refers to the smallest relative error one obtains when initializing and optimizing a network 20 times.

In Fig. 5.29a one can set that TRT achieves great results with very little parameters, but this does not completely translate to the alternating sign case as Fig. 5.29b shows. The smallest successful configuration from Fig. 5.29b is $n^{(1)} = n^{(2)} = 8$, which corresponds to a total of $|\theta| = 136$ parameters, which is 10 parameters less than the TD network needs. This is not much better but remember that we do not exploit any symmetry yet.

Recall from Sec. 5.2 that the case $N = 8$ yields a symmetric ground-state wave-function. For $N = 6$, accordingly we obtain an anti-symmetric wave-function. Even though the network can find a representation at machine precision it needs nearly the maximally allowed number of parameters to do so, as one can see in Fig. 5.30.

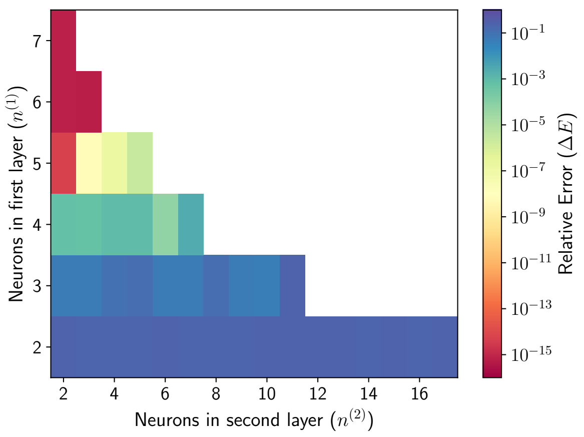
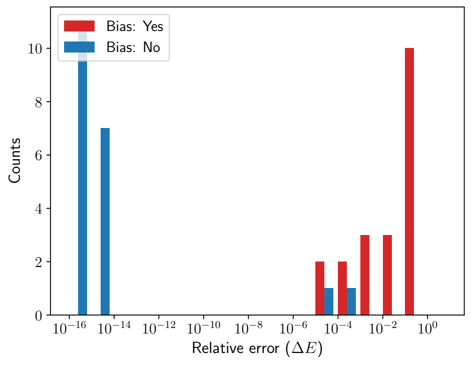


Figure 5.30: Model: TRT, Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 6$

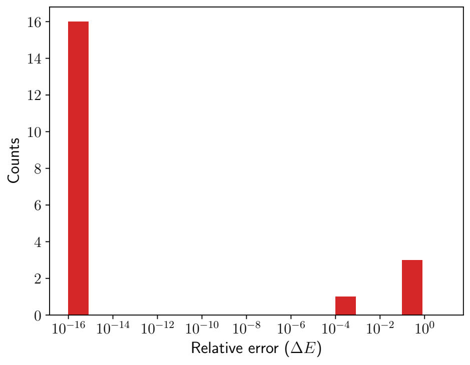
Anyway, it is still intriguing that a three-layer network with a strong gradient is able to achieve such results. The TD network is built specifically to exploit the symmetry and to work on both symmetry cases but outperforms TRT only slightly on the anti-symmetric case. The next step is therefore to find an architecture that combines the training advantage of TRT with the symmetry of the TD architecture.

5.5.2. Tanh - Triangle - Tanh

The T is an anti-symmetric function, thus it conserves the symmetry of any function that it is composed with. The TD architecture combines the gradient advantage from R and the symmetry advantage from C. For the anti-symmetric case the C function becomes a sinus, and for values in $(-\pi, \pi)$ the sinus behaves very similar to the T. Thus, the test from Fig. 5.17b gives quite a good estimation for what one might expect when using TD as ansatz. One can obtain a high precision but not with such great reliability as TRT (see 5.21a) yields on $N = 8$. Thus, the idea is now to use the fact that D has similar gradient properties like R but has the advantage of being either symmetric or anti-symmetric. So, when we sandwich D between two T layers, we can determine the right symmetry in the second layer and the output layer will retain it, due its anti-symmetric character, no matter which symmetry it is. Hence, we introduce the ansatz TDT from Fig. 5.31. To verify that we have the same performance capabilities with TDT as with TRT, we test the same way as in Sec. 5.2, of course the test is carried out for $N = 8$ on which TRT performs best.



(a) Model: TRT



(b) Model: TDT

Figure 5.32: Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 8$

As it turns in Fig. 5.32 out TDT is even more successful in finding a solution at machine precision then TRT. Now we are of course interested in a case of an anti-symmetric wave-function Eq. (5.6), which can be investigated on a system of size $N = 10$. With the TTT architecture it happens to be such that the architecture does not perform well on the positive sign case (Fig. 5.4b) but is capable of very precise approximations for the anti-symmetric mixed sign case (Fig. 5.18a). To assure ourselves that TDT preforms on all cases the same we look now at the system size $N = 10$ for Hamiltonian $\mathcal{H}_{ps}$ and$\mathcal{H}_{ms}$ in Fig. 5.33.

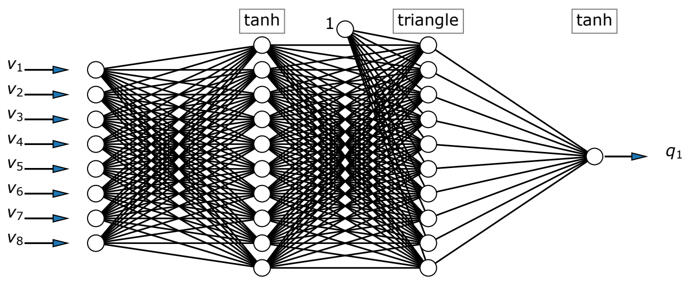
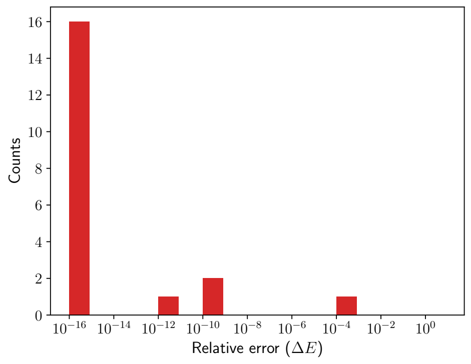
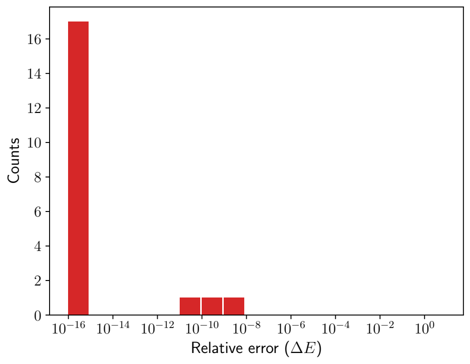


Figure 5.31: The picture shows an example for a three-layer network, composed two T and a D activation functions. The bias is only active for the D layer.



(a) Hamiltonian: $\mathcal{H}_{ms}$



(b) Hamiltonian: $\mathcal{H}_{ps}$

Figure 5.33: Model: TDT, System size: $N = 10$

As Fig. 5.33b and Fig. 5.33a show TDTs success of finding a representation of the ground-state at machine precision does not depend on whether we have a positive definite wave-function or an anti-symmetric one. Actually, TDT performs great on every symmetry case we test it on. Now one may ask why TDT is so much more successful in finding a good representation than TD. The basic concept of TD and TDT is very similar. They both rely on finding the correct bias in the layer with the periodic function which is equivalent to learn the right symmetry of the wave-function. The answer to this lies in Fig. 5.34.

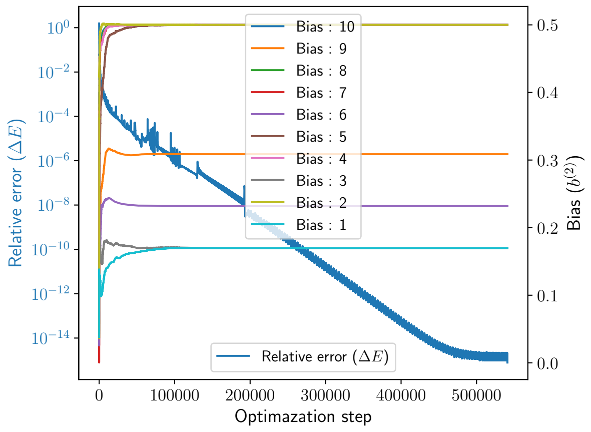
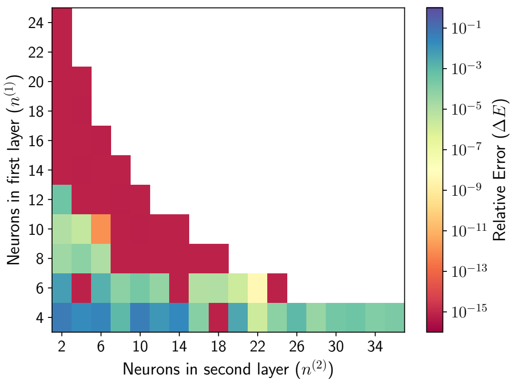


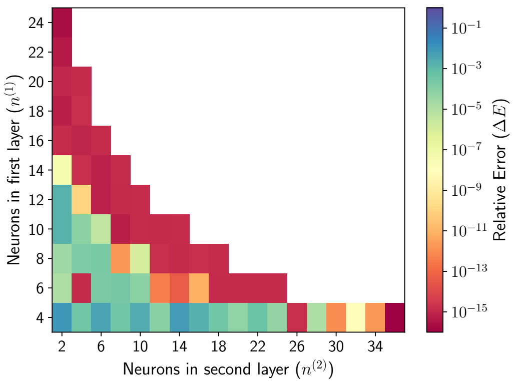
Figure 5.34: With the TDT architecture of homogeneous layer-size and parameters number $\mathcal{D}$ we search the ground-state of Hamiltonian $\mathcal{H}_{ms}$ with system size $N = 10$ (while we restrict the configuration space to the subspace of $m_{\mathbf{v}} = 0$). In this figure one can see the evolution of the bias in the network during the optimization process, such as the relative error's evolution.

The main difference between TD and TDT is, that instead of just one bias in

TD, the network TDT has to find the right bias for all the neurons that are in the second layer. In an ideal case for an anti-symmetric function, like for $N = 10$, all the biases of the second layer would be found to be $b_j^{(2)} = 0.5$, such that the triangle function (Fig. 5.12) has an anti-symmetric character. In figure Fig. 5.34 one can see the evolution of the bias during a training procedure that yields a maximally precise approximation of the ground-state energy. The point we want to make with this is that it turns out that the assumption, all biases must have the right value, is wrong. Fig. 5.34 shows that even when 4 out of 10 biases have not the right value, the network manages to yield machine precision anyway. Thus, the advantage of TDT over TD is the introduced redundancy, that yields a way higher rate of success. The question we want to answer now, is if this redundancy comes at the cost of more required parameters.



(a) Hamiltonian: $\mathcal{H}_{ms}$



(b) Hamiltonian: $\mathcal{H}_{ps}$

Figure 5.35: Model: TDT, System size: $N = 8$

The answer, that Fig. 5.35 provides, is that in the case of Hamiltonian $\mathcal{H}_{ms}$ and $\mathcal{H}_{ps}$ for $N = 8$, TDT requires about half the number of parameters that TD requires in Fig. 5.28a to yield machine precision. The following numbers are the smallest number of parameters that the associated networks require to yield machine precision for Hamiltonian $\mathcal{H}_{ms}$ and $N = 8$

$$\text{TD} : |\theta| = 145 \tag{5.14}$$
$$\text{TRT} : |\theta| = 136 \tag{5.15}$$
$$\text{TDT} : |\theta| = 80 \ . \tag{5.16}$$

Thus, the redundancy does not lead to more required parameters, but actually requires less. This is in great correspondence to the advantage of biological neural networks. Their power does also come from exploiting parallelism and redundancy, as we discuss in Sec. 4. To make a last comparison to the TRT network we check what the smallest number of parameters is, with which TDT is capable of yielding machine precision for $\mathcal{H}_{ms}$ at $N = 6$.

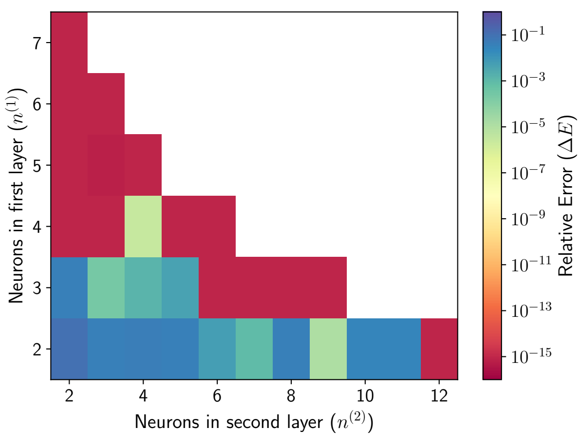


Figure 5.36: Model: TDT, Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 6$

The result that Fig. 5.36 provides tells us that, while we need at least $|\theta| = 50$ parameters with the TRT network, TDT yields a computational exact ground-state energy with just $|\theta| = 40$. But keep in mind that small systems sizes are not so expressive as larger systems. Like the fact that TRT is not capable of yielding machine precision at all, in the case of $N = 10$ ($\mathcal{H}_{ms}$), is more meaningful.

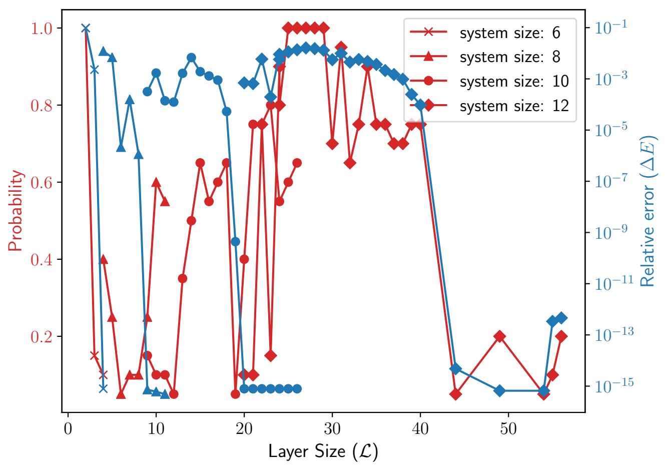


Figure 5.37: In this figure one can see the precision that the TDT network yields at best for a homogeneous layer-size on the x-axis. In red one can see the probability with which the corresponding probability is obtained. Unfortunately, we cannot provide the full data for the range from $M = 38$ to $M = 53$.

When we compare the histograms of precision and reliability to the plots that show the smallest possible layer-sizes, we cannot cope with the feeling that it would be useful to know with which rate we get the precision that is associated to a certain layer configuration. This is what Fig. 5.31 shows. The behavior that Fig. 5.37 illustrates is very much as one would expect. There is a threshold the layer-size has to surpass after which the network is capable of yielding machine precision. Close to the threshold the probability to get to machine precision is small but increases as the layer-size increases. What Fig. 5.37 further reveals is that for $N > 6$ the required number of neurons is about $\mathcal{D}/2$. If this holds true for larger systems, this network is not feasible (at least for equally large layers, the parameter growth is linear when it is enough to grow only the first layer). But this investigation exceeds the scope of this work. Instead we proceed in this discussion of the TDT network with evaluating the spin-correlation function. The spin-correlation function quantifies how strong two spins on the lattice couple two one another [34]

$$\Gamma^{zz}_{ij} = \langle S^z_i S^z_j \rangle_q \ . \tag{5.17}$$

By evaluating the correlation function for the variational q that TDT yields and comparing to the q obtained from Exact Diagonalization we can determine how precise the approximation of the wave-function is.

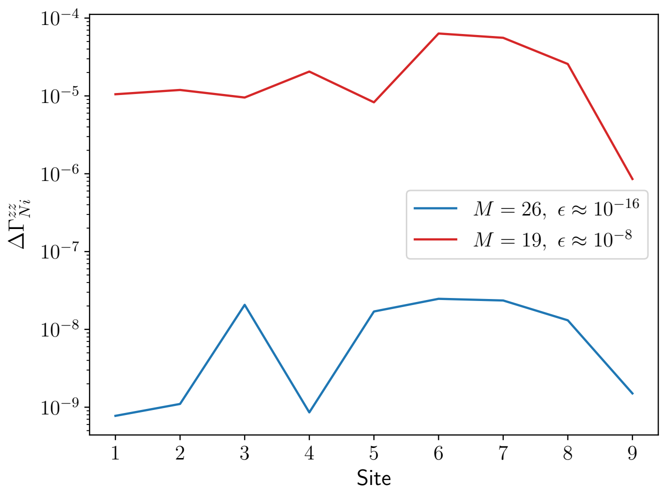
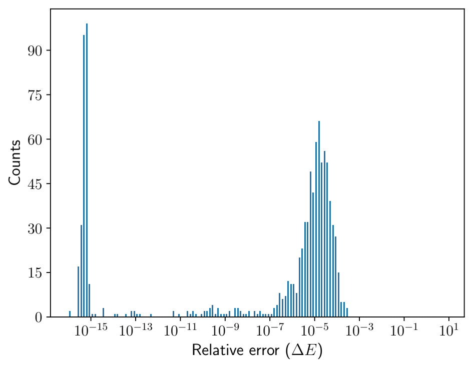


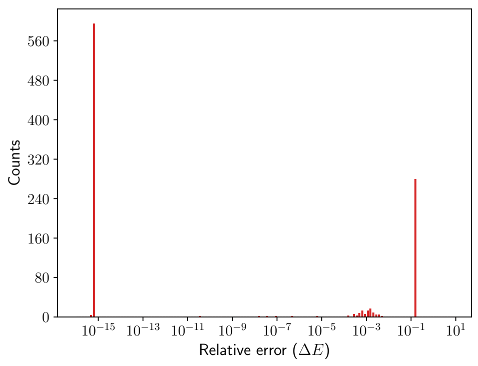
Figure 5.38: Here one can see the correlation function of the variational wave-function for the ground-state of Hamiltonian $\mathcal{H}_{ms}$ $N = 10$ compared to the result from Exact Diagonalization $\Delta\Gamma^{zz}_{0i} = |(\langle S^z_N S^z_i \rangle_{var} - \langle S^z_N S^z_i \rangle_{ed})/\langle S^z_N S^z_i \rangle_{ed}|$. In the red case we employ a network that has first and second layer-size set to $M = 19$ and yields a precision of $\Delta E \approx 10^{-8}$. The blue plot line is the case of $M = 26$, which yields precision $\Delta E \approx 10^{-16}$.

So far, we have only looked at how precisely we approximate the ground-state energy and not how good the variational wave-function resembles the actual wave-function. Thus we evaluate the correlation function between every pair of sites ($i = N$ and $j \in \{1, 2, ..., N-1\}$), the result is shown in Fig. 5.38 for Hamiltonian $\mathcal{H}_{ms}$ $N = 10$. Thereby we compare the correlation function that we calculate with a network of $M = 19$ and one of $M = 26$ neurons in the first and second layer. The smaller network is not trained to yield machine precision, but such that the training converges to an approximation that yields $\Delta E \approx 10^{-8}$. The larger network yields $\Delta E \approx 10^{-16}$ and its correlation function has a relative error of about 10^{-8}. When we compare this to the smaller, less precise network we find that the correlation function differs by about 10^{-4} of relative error from the actual wave-function. This puts the relative energy error and quality of the wave-function into perspective

Finally, we want to conclude this section with the reiteration of test Fig. 5.4b and Fig. 5.32b, but this time we use 1000 optimization results to make histograms with eight times higher resolution. With the results in Fig. 5.39 we want to emphasize that the previous histograms, that contain only 20 optimization attempts, provide nearly the same information as 1000 do.



(a) Model: TRT, Hamiltonian: $\mathcal{H}_{ps}$



(b) Model: TDT, Hamiltonian: $\mathcal{H}_{ms}$

Figure 5.39: System size: $N = 8$, Number of optimization attempts: 1000

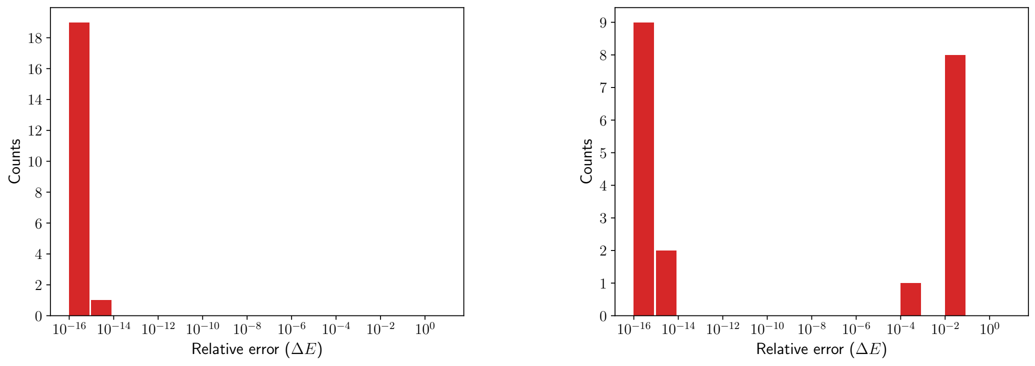
Though some additional interpretation can be made, for the TDT network we find the clear minima. One of which is of course at the desired $\Delta E \approx 10^{-16}$ and the other one is at $\Delta E \approx 10^{-1}$. The letter minimum is found when the biases are not determined right. Beyond these two obvious cases there is one smaller minimum at $\Delta E \approx 10^{-3}$, which seems to be pretty shallow as the optimization gets not often trapped in there. For the TRT network we find in Fig. 5.39a that around $\Delta E \approx 10^{-5}$ there seem to be at least one broad and deep minimum in which the optimization gets trapped most of the time. Nonetheless is this information not of particular use and does not lead to other conclusions than the ones we draw earlier on.

5.5.3. Shallow Network versus Deep Network

When we recall the TD network from Sec. 5.4.2 we find that its biggest drawback is the low rate of training-success in the case of an anti-symmetric wave-function. The root of this struggle is the need to determine the bias in the output-layer that determines the symmetry of the network. This procedure is very cumbersome, but we overcome this problem in Sec. 5.5.2 by introducing redundant neurons that all determine a bias and thus the network is not dependent from the success of a single neuron. In the case of TDT we introduce an additional layer to create redundant neurons, but when we get rid of the first layer we obtain a two-layer network that also has redundant biases

$$h_{W,b}(\mathbf{v}) = \tanh(\mathbf{w}^{(2)} \cdot \mathbf{D}(\mathbf{W}^{(1)}\mathbf{v} + b^{(1)})) \tag{5.18}$$

the abbreviation of the resulting network is DT. So let us repeat the reliability tests now with DT.



(a) System size: $N = 8$, Number of optimization attempts: 20

(b) System size: $N = 10$, Number of optimization attempts: 20

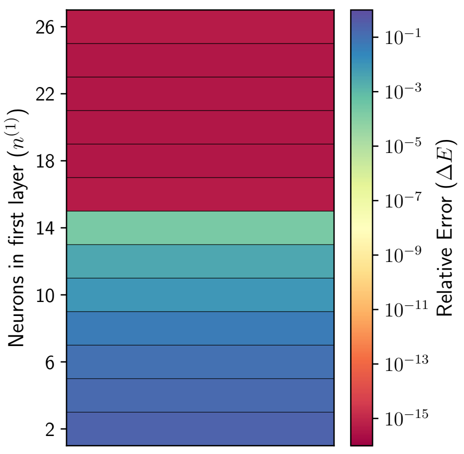
Figure 5.40: Model: DT, Hamiltonian: $\mathcal{H}_{ms}$

In Fig. 5.40 one can clearly see the advantage arising from redundancy when comparing to the results from Fig. 5.27, for completeness we sum up the reliability results in Tab. 3. There we calculate the probabilities to obtain computationally exact ground-state energies when optimizing the associated networks.

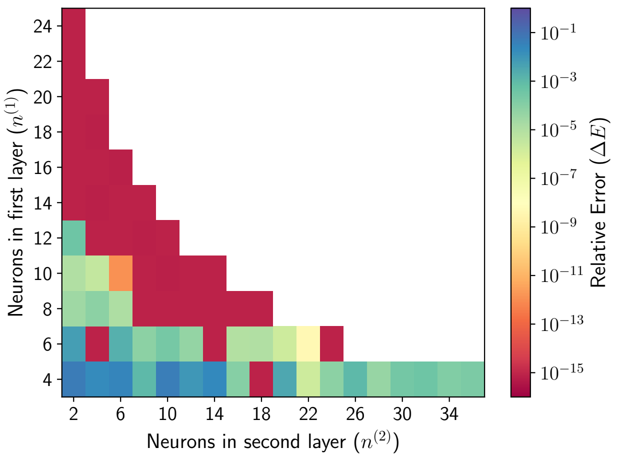
System size \ Network	TD	DT	TDT
$N = 8$	0.3	0.95	0.8
$N = 10$	0.04	0.45	0.8

Table 3: Here we compare the probability to obtain computationally exact ground-state energies from optimizing three different network architectures for Hamiltonian $\mathcal{H}_{ms}$. The results are calculated from Fig. 5.40, Fig. 5.27, Fig. 5.33a and Fig. 5.32b.

Tab. 3 shows clearly that the networks with redundant biases (DT and TDT) are way more reliable than the TD, that has no redundancy. Further we see in Tab. 3 that the DT preforms slightly better in the symmetric case than TDT, but TDT performs significantly better in the anti-symmetric case. Though this is not yet the whole reason why TDT is actually an improvement over the DT network. In Sec. 4.2 we argue that there are functions, that a three-layer network can efficiently approximate to certain precision, and a two-layer network can accomplish the same precision only with at least exponentially many parameters. Thus, we now compare the number of parameters that are required by the DT and the TDT network to yield computationally exact ground-state energies.



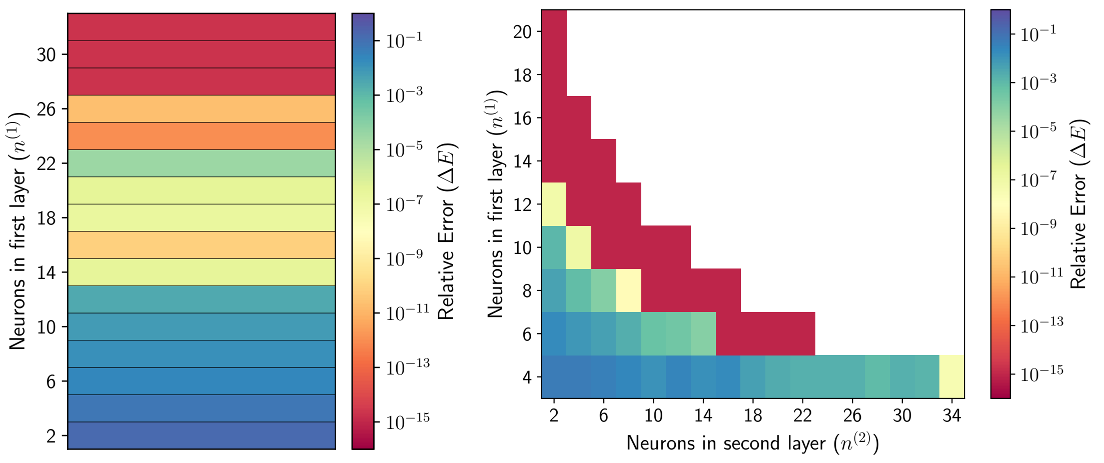
(a) Model: DT



(b) Model: TDT

Figure 5.41: Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 8$, Training space: $\mathcal{T}$

In Fig. 5.41 we have the test results for Hamiltonian $\mathcal{H}_{ms}$ and $N = 8$, while in Fig. 5.42 we have $N = 10$ but restricted to the $\mathcal{T}_0$ training set, this way we can cover both symmetry cases. To yield a computationally exact ground-state energy for the case Hamiltonian $\mathcal{H}_{ms}$ and $N = 8$ the TD network requires $n^{(1)} = 16$ neurons in the first layer, which is the same amount that DT requires. So other than the reliability, the number of neurons shows no different behavior for TD than it does for DT.



(a) Model: DT **(b)** Model: TDT

Figure 5.42: Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 10$, Training space: $\mathcal{T}_0$

In Tab. 4 one can find the result of the comparison of minimal number of parameters for the networks TD, DT and TDT. From Tab. 4 we learn that DT requires the most parameters of all networks to do the approximation for $N = 8$, this is due to the additional biases that TD not has. The least parameters are required by TDT, which needs half of the amount that DT requires. In the case of $N = 10$ the DT network is not able to do the approximation to the demanded precision with $|\theta| < \mathcal{D}$, while the TDT network does it with $|\theta| = 172$. Hence, we can confirm the advantage of deep networks over shallow networks for this problem. But let us stress for a moment that DT succeeds very well at finding an approximation of the ground-state wave-function that yields a computationally exact eigenvalue for $N = 10$ when we train it on $\mathcal{T}$ (Fig. 5.40b). When we aim to do the same on the training set $\mathcal{T}_0$ we fail for $|\theta| \leq \mathcal{D}$, where in this case $\mathcal{D} = 252$. Restricting the ground-state wave-function to $\mathcal{T}_0$ means that we leave out all configurations that

System size \ Network	TD	DT	TDT
$N = 8,\ \mathcal{T}$	145	160	80
$N = 10,\ \mathcal{T}_0$		336	172

Table 4: Here we compare the minimal number of parameters ($|\theta|$) that are required to obtain computationally exact ground-state energies from optimizing three different network architectures for Hamiltonian $\mathcal{H}_{ms}$. The results are calculated from Fig. 5.28a, Fig. 5.41 and Fig. 5.42.

are mapped to $q(\mathcal{T} \setminus \mathcal{T}_0) = 0$ by the exact wave-function q, thus the network has to learn the non-trivial part of the wave-function using at most $|\theta| = 252$ instead of $|\theta| = 1024$ parameters. While the three-layer TDT network manages to find a solution that is in the boundary of maximal allowed parameters, the two-layer DT network does not. Eventually we find that redundant neurons and functions with strong gradients help increasing the training success, while depth allows to decrease the required number of parameters for a certain approximation. Though too much depth causes loss of reliability, as we find in Fig. 5.23 and Fig. 5.12 for networks that contain an R activation function.

5.6. Test Conclusion

In the introduction to this work we say that in Machine Learning one has heuristic rules at his disposal that help building a network for a specific task [24] and we want to derive something similar for the problem of the quantum ground-state. Now that we are finished carrying out the tests that concern the architecture and drawing conclusions from these tests we want to cast these conclusions into our desired guidelines. In Sec. 5.2 - Sec. 5.5 we go from a general approach for the ground-state to a neural network that is actually capable of reliably yielding computationally exact ground-state energies, this development contains already all the guidelines, here we just mean to state them again explicitly.

The first guideline we state is concerning *depth*. We observe very early in Sec. 5.2 that depth increases the expressiveness of a network but increases also the training complexity. Though in Sec. 5.5 depth helps reducing significantly the parameters we need for a computationally exact energy. Thus, we propose a tempered use of depth:

1. A shallow network is favorable respecting the gradient descent optimization with back-propagation. Though, a deeper network might give an advantage in terms of parameter efficiency. We find that already three-layer networks show this effect and this while not suffering too much from training problems, thus we recommend a network depth of $2 < \mathcal{L} < 5$.

We also realize very early on that functions with strong gradients help increasing the training success. In Sec. 5.2 we make this observation and in Sec. 5.4 realize that if we retain the qualitative character of a function while making it rougher (to obtain a function with stronger gradient), we preserve the expressiveness of a network while increasing its training success. This is reflected by the second guideline:

2. Employ functions with strong gradients. If the function you want, or need, to use does not have a strong gradient try to approximate it with a function that has (the activation functions do not need to be smoothly differentiable).

When we move on in Sec. 5.3 to investigate Hamiltonian $\mathcal{H}_{ms}$ we are confronted with the problem of two different kinds of symmetry for $N = 8$ and $N = 10$. There we can observe that networks are biased towards symmetry. Even when activation functions do not appear to have an intrinsic symmetry (like R and S) we find that they may perform on one symmetry case better than on the other. We resolve this in Sec. 5.4 by introducing functions that have symmetries, but their symmetry character can be changed using *bias*. Hence, we recommend:

3. Be sure that the activation functions support the symmetry properties of the desired wave-function. If no symmetry is known use activation functions that do allow for any symmetry.

The last guideline is connected to the third guideline, because following that guideline alone does lead to better performance but not necessarily to higher reliability. Recall the case of the networks TD and DT (see Sec. 5.5.3 for the test results), which show both the same precision, but DT has the much higher training success rate than TD. This is due to the redundant bias in DT, that makes it independent of the right determination of a single bias, while TD depends on a single bias. So eventually we recommend employing redundancy:

4. If there are crucial values to be determined in a layer, add at least one layer after that one. This way you can make the crucial layer of a sufficiently large size, such that redundancy allows to be independent from the success of a single neuron.

These guidelines are of course very general and do not exclusively apply to the ground-state problem, they are more a direct result of tackling this specific problem. In the section that succeed this part we discuss some ideas that are more specific to finding the ground-state wave-function but do not contribute to the guidelines.

6. Alternatives to the Forward-Propagation

In Sec. 4.1 we introduce the linear function Eq. (4.5), that is employed in neural networks, to process information. This function can be changed with the goal to save parameters or accelerate training. In the following section we investigate two alternatives for the standard linear function.

6.1. Sparse Connection

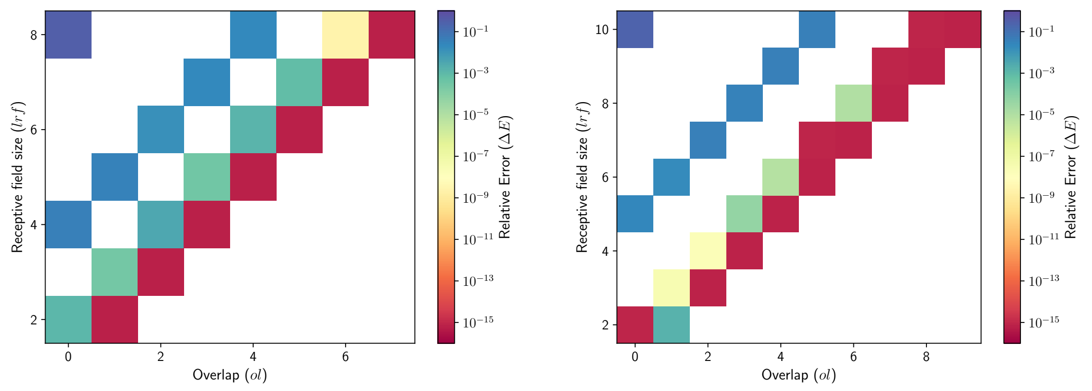
In section Sec. 7.2 we introduce the term *k-local* many-particle Hamiltonian. Let us state more clearly what that means. When a Hamiltonian is k-local it mediates interactions between k particles. Thus, every interaction-term, that the Hamiltonian $\mathcal{H}$ contains, can be seen as a local Hamiltonian h_i that acts only on k particles. The total Hamiltonian can be decomposed into the sum of local Hamiltonians

$$\mathcal{H} = \sum_i h_i \, . \tag{6.1}$$

So the information of the wave-function is encoded in groups of k sites, in terms of a spin chain. Let us consider the scenario in the first layer. The linear operation Eq. (4.5) makes a weighted sum of the orientation of every single spin on the lattice. But instead of doing that, we try an approach, where we make a weighted sum of $k \leq N$ sites. We call k, the number of interacting sites in terms of the network input, *local receptive field* (lrf). And the amount of overlap of the single receptive fields is denoted *overlap* (ol). The linear function Eq. (4.5) reduces to the *sparse linear function*

$$z_i^{(1)} = \sum_j^{lrf^{(l)}} W_{ij}^{(0)} v_{j+i(lrf^{(l)}-ol^{(l)})} + \mathbf{b}^{(0)} \, , \tag{6.2}$$

(see App. B for the implementation in the back-propagation algorithm) where every node is a superposition of only lrf-many input nodes and neighboring nodes have ol-many common input nodes. The sparse linear function, Eq. (6.2), allows to coarse-grain the input vector $\mathbf{v}$ and has thus, loose similarities to the *Renormalization Group* (RG) technique. For the *Deep Boltzmann Machines* from Sec. 7.1 a connection to RG is actually known [56], but here we have other intentions. To find the wave-function amplitude associated to a configuration $\mathbf{v}$ the neural network has to extract features from this configuration. The features are represented in the values of the successive neurons but might be extracted *locally* and allow for a sparse connection between neutrons, instead of a dense connection. Ideally this leads to a better scaling of the network parameters with respect to the system size.



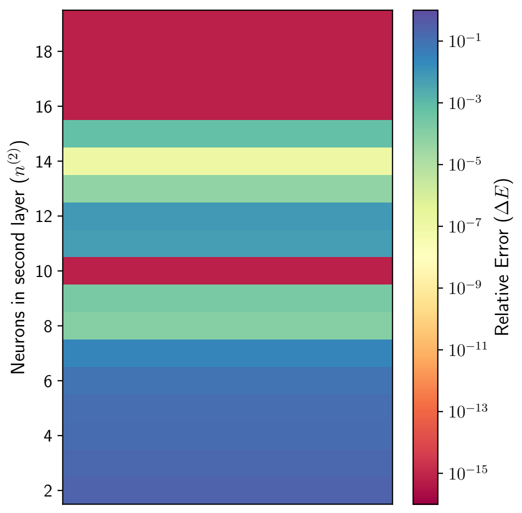
(a) System size: $N = 8$, Training space: $\mathcal{T}$ **(b)** System size: $N = 10$, Training space: $\mathcal{T}_0$

Figure 6.1: Model: sparse TDT, Hamiltonian: $\mathcal{H}_{ms}$; In this test we change the sparsity of the first layer and adapt the second layer-size in accordance to Eq. (6.3).

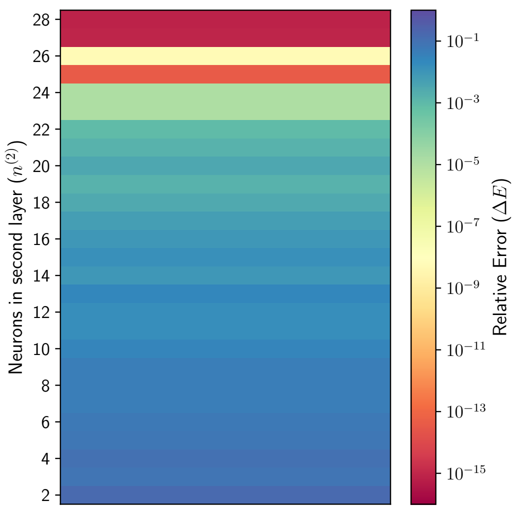
To test this approach, we try all possible combinations of lrf and ol. Doing so we set the configuration of the first layer, but the size of the second layer has to be calculated. We calculate it in such a way that the total number of parameters is $\mathcal{D}$, if the first layer were a fully connected layer

$$\frac{\mathcal{D} - N/(lrf - ol)}{N/(lrf - ol) + 2} = n^{(2)} , \tag{6.3}$$

where we allow only those lrf and ol for which $N/(lrf - ol) \in \mathbb{N}$. We investigate Hamiltonian $\mathcal{H}_{ms}$, which is a 2-local Hamiltonian. We test system sizes $N = 8$ and $N = 10$ (where we restrict $\mathcal{T}$ to $\mathcal{T}_0$ for $N = 10$), to cover both symmetry cases. We implement the sparse linear function with the TDT network. Fig. 6.1 shows that we get machine precision in the cases when $ol = lrf - 1$, meaning that the overlap is maximal. Only in Fig. 6.1b, the anti-symmetry case, we find that $ol = lrf - 2$ yields machine precision (only for $lrf = 2, 7$). Having sorted out that the network is yielding machine precision this way, we want to know what the minimal $n^{(2)}$ is to still get machine precision. We initialize the network 20 times randomly and count how many times we achieve machine precision in doing so, since we are also interested in how reliable the network is.



(a) System size: $N = 8$, Training space: $\mathcal{T}$

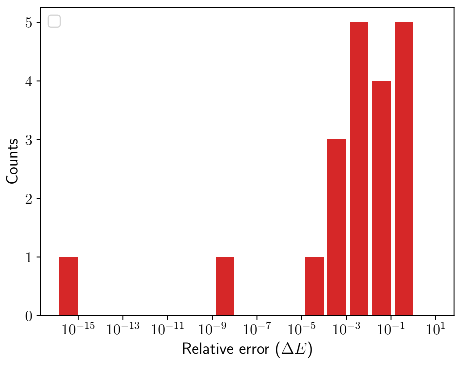


(b) System size: $N = 10$, Training space: $\mathcal{T}_0$

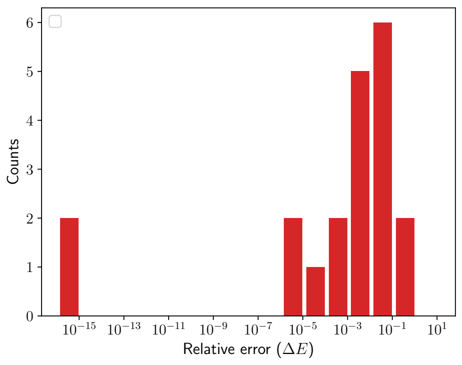
Figure 6.2: Model: sparse TDT, Hamiltonian: $\mathcal{H}_{ms}$; Here we try to find the minimal $n^{(2)}$ that allows to get to machine precision. In Fig. 6.2a we have $lrf = 2$ and $ol = 1$, while in Fig. 6.2b we set $lrf = 2$ and $ol = 0$.

Fig. 6.2a shows that machine precision can be achieved with $n^{(2)} = 10$ which corresponds to a total number of parameters of $|\theta| = 112$ which stands against the minimal number of parameters $|\theta| = 80$ from Fig. 5.32b. Thus, we have no advantage in terms of parameters. When we consider that in Fig. 6.2a $n^{(2)} = 11 - 15$ do not yield machine precision when can conclude that the probability to obtain machine precision for $n^{(2)} = 10$ is very low. For the case $n^{(2)} = 19$ we have the precision-counts histogram in Fig. 6.3a and there we can see that for the maximally allowed number of parameters only 4 out of 20 optimization attempts let obtain machine precision. When we move on to the anti-symmetric wave-function (Fig. 6.2b) we find that the smallest successful layer is $n^{(2)} = 27$ which corresponds to $|\theta| = 334$. To have a comparison we test now what the smallest number of parameters is, that can be achieved with a fully connected TDT network. The result, that we get from figure Fig. 5.42b, is $|\theta| = 172$ and is hence about half of the total parameters we need with a local receptive field ($lrf = 2$ and $ol = 0$) in the first layer. Also, the probability to obtain machine precision is only 2 out of 20 (for this result see Fig. 6.3b).

The idea in using local receptive fields is to exploit the fact that most physical Hamiltonians can be decomposed in a sum of local k-body Hamiltonians and thereby by save parameters or make learning more reliable. But it turns that this *local* character of physical states does not translate to the network weights, at least in terms of efficiency. The best approach is still the standard fully connected layer.



(a) System size: $N = 8$, Training space: $\mathcal{T}$



(b) System size: $N = 10$, Training space: $\mathcal{T}_0$

Figure 6.3: Model: sparse TDT, Hamiltonian: $\mathcal{H}_{ms}$

6.2. Convolution

Convolutional Neural Networks (CNN) are Artificial Neural Networks that use a *discrete convolution* operation instead of the standard matrix vector multiplication (Eq. (4.5)). CNNs are hugely successful in performing tasks like image classification and localization [12, 6]. Classification is a discriminative task and is not directly related to the problem of approximating very precisely a specific function. Nonetheless, we are interested in investigating whether we can use this technique to reduce the dimension of the Hilbert space or not. Hamiltonian $\mathcal{H}_{ms}$ and $\mathcal{H}_{ps}$ are transnational invariant. This means that Hamiltonian $\mathcal{H}_{ms}$ (such as $\mathcal{H}_{ps}$) acts the same on $|\mathbf{v}\rangle$ as on $|\mathbf{v}'\rangle$ when there is a j such that

$$\mathbf{v}' = \sum_{i=1}^{N} v_i \mathbf{e}_{(i+j) \mathrm{mod} N} \ , \tag{6.4}$$

where $\{\mathbf{e}_i\}$ is the standard basis in N dimensions. Thus, we can reduce the complexity of finding the ground-state when the network treats configurations, that can be transformed into one another by translation, the same

$$h_{W,b}(\mathbf{v}) = h_{W,b}(\mathbf{v}') \ . \tag{6.5}$$

What we do to achieve this is to introduce a second index, as it is done in conventional CNNs [18]

$$z_{lj}^{(i+1)} = \sum_{h}^{n^{(i)}} W_{lh}^{(i)} a_{h+j}^{(i)} + b_j^{(i)} \ . \tag{6.6}$$

The translational index in the activation tensor in Eq. (6.6) comes from the implementation of discrete convolutions

$$(\mathbf{w} * \mathbf{a})(j) = \sum_{h=1}^{n} w_h a_{(h+j) \mathrm{mod}\ n} \ , \tag{6.7}$$

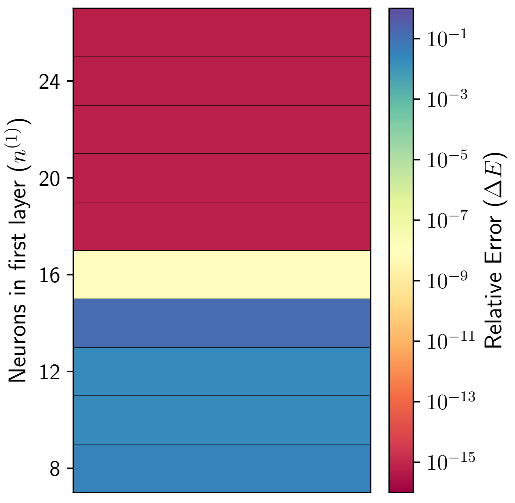
where the *kernel* $\mathbf{w}$ is a tensor with only one index. In conventional convolution layer the two-times indexed tensor $z_{lj}^{(i)}$ is now propagated through the network and the weights become three-times indexed tensors $W_{lhk}^{(i)}$ [4]. Therefore, every translation k leads to its own weight matrix $W_{lh}^{(i)}$ and implements not really the transnational invariance we desire, but instead introduces more parameters. So, to propagate the activation through the network we need to trace over one index in Eq. (6.6), in order to obtain a vector again. But we are given two options to do so. The first option we have is to trace over l, which yields

$$z_j^{(i+1)} = \sum_{l}^{k} \left(\sum_{h}^{n^{(i)}} W_{lh}^{(i)} a_{h+j}^{(i)} \right) + b_j^{(i)} \ . \tag{6.8}$$

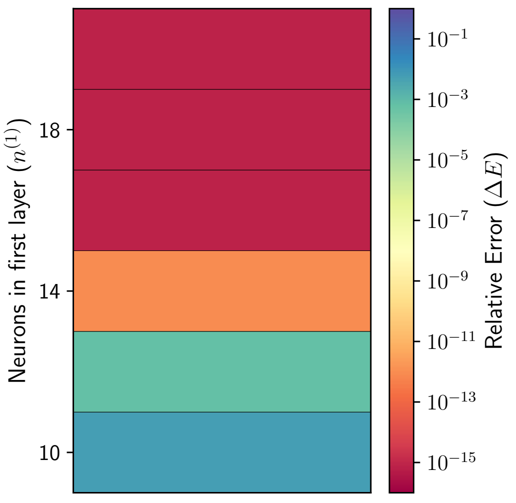
Eq. (6.8) enforces $\mathbf{W}^{(i)} \in \mathbb{R}^{k \times n^{(i)}}$ and $n^{(i)} = n^{(i+1)}$, where k is an arbitrary integer. The result of tracing over l is not translational invariant and cannot be used as output layer since this requires $n^{(i+1)} = 1$. Therefore, we move on to trace over l, which leads to

$$z_j^{(i+1)} = \sum_{l}^{n^{(i)}} \sum_{h}^{k} W_{jh}^{(i)} a_{h+l}^{(i)} + b_j^{(i)} = \left[w_j^{(i)} = \sum_{h}^{k} W_{jh}^{(i)} \right] = w_j^{(i)} \sum_{l}^{n^{(i)}} a_l^{(i)} + b_j^{(i)} \ , \tag{6.9}$$

(see App. B for the implementation in the back-propagation algorithm). In Eq. (6.9) $k < n^{(i)}$ is an integer, but as $\mathbf{W}^{(i)}$ can be reduced to a vector $\mathbf{w}^{(i)} \in \mathbb{R}^{n^{(i+1)}}$ the number k does not matter anymore. It is obvious that Eq. (6.9) is a scaled sum of all entries of $a_l^{(i)}$ plus a offset. Thus, it is translational invariant, but there is no need to have more than one node. Other than having a different offset and scaling on every node we cannot extract any additional information by increasing the number of nodes. An even more serious problem arises when the convolution is in the first layer. Not all configuration vectors in $\mathcal{T}$, that have the same sum of entries, are connected by a translation. Thus, the convolution cannot be in the first layer and is best positioned in the output layer. To overcome the problem of degeneracy, in terms of the sum of configuration vector entries, we need to transform the problem from the N-dimensional input space to a higher dimensional space. The idea is that we use a TD two-layer network, where the linear output function is Eq. (6.9) and the input layer is fully connected. In this network just the size of the first layer has to be determined. Thus, we consecutively increase the layer-size and see what the precision is that we can get out of 20 independent optimization processes.



(a) System size: $N = 8$, Training space: $\mathcal{T}$



(b) System size: $N = 10$, Training space: $\mathcal{T}_0$

Figure 6.4: Model: convolutional TD, Hamiltonian: $\mathcal{H}_{ms}$

We test on Hamiltonian $\mathcal{H}_{ms}$ with system size $N = 8$ (on the full configuration set $\mathcal{T}$) and $N = 10$ (on the reduced configuration set $\mathcal{T}_0$) to cover both symmetry cases (see Sec. 5.3). In figure Fig. 6.4a we find that $|\theta| = 146$ parameters are required for machine precision when $N = 8$ and $|\theta| = 162$ parameters are required according to Fig. 6.4b for $N = 10$. In terms of $N = 8$ the convolution network requires about double the number of parameters that the fully connected TDT requires (Fig. 5.32b). But in the case of $N = 10$ the convolutional TD outperforms TDT by 11 parameters (Fig. 5.42b). The fact that we do so implies that in the convolutional TD we are able to exploit spin-inversion symmetry and translational symmetry of the Heisenberg Hamiltonian, while TDT exploits only the spin-inversion symmetry. But against this conclusion stands the fact that TDT outperforms the convolutional network in the symmetric case $N = 8$. Thus, one has to take this result with care. The convolutional approach can be an advantage, but it is not the most versatile approach. This gets clear when we stress again that the sum of vector entries allows just for a scaling factor as the only parameter in the output. Therefore, a convolutional TD network is simply a special case of the TD network, where the output layer has the same entry in every component of the output matrix. Hence the convolution reduces the training complexity of the TD network, but this does not loosen the bounds we find in Sec. 5.4.2 for the

TD network.

Eventually is the change of the linear function merely an additional degree of freedom in the design of a neural network. But other than by tweaking depth and activation functions we do not find any advantage in deviating from the standard linear function.

7. Boltzmann Machines

So far, we have only treated networks that take a certain input vector, process it in a consecutive manner and return an output vector. Now we look at a type of network, that is *undirected* and *probabilistic*. This kind of network takes an input vector as we know it, but the processing is not done by propagating the vector through the network. The nodes of a probabilistic network correspond to random variables, and processing a vector corresponds to evaluating the conditional probabilities between the input nodes and all other nodes. In this section we give the definition of probabilistic models and show a way to use them to better train feed forward networks.

7.1. Restricted Boltzmann Machines

A *Boltzmann Machines* (BM) is a form of network, that is conceptually different from what we establish in Sec. 4. BMs and ANNs have very similar graphical representations. But while for ANNs the representation as graph, like in Fig. 4.4, has merely an explanatory purpose BMs have a fundamental relation to their representative graph. In fact, a BM is a special case of what is called a *probabilistic graphical model*. Probabilistic graphical models are probability distributions that describe the joint distribution of a set of random variables $\{X\}$.

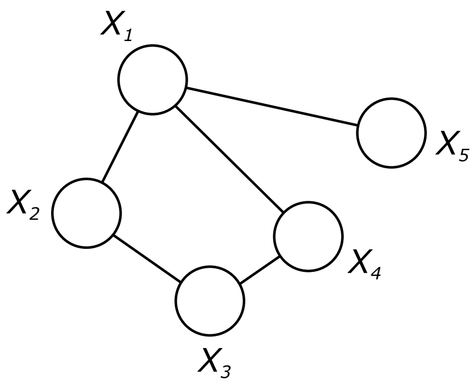


Figure 7.1: This is an example for a graphical model. Every node is associated to a random variable. Edges between nodes define conditional dependencies between random variables.

Considering a graph like Fig. 7.1 every random variable $\{X\}$ is associated to a node and the edges define conditional dependencies. As an example take X_1, X_3 and X_4 then their conditional probability distribution factorizes as

$$P(X_1, X_3|X_4) = P(X_1|X_4)P(X_3|X_4) \ . \tag{7.1}$$

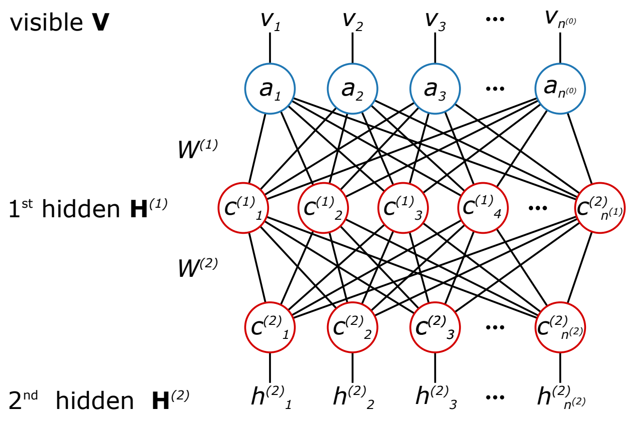
In terms of the BM we make a distinction between random variables $\mathbf{V} = \{V\}$ that are *observed* and random variables $\mathbf{H} = \{H\}$ that are not observed. Both types of random variables are binary and take values in $v, h \in \{-1, 1\}$. We refer to $\mathbf{V}$ as an observable random variable, as it is the random variable that corresponds to an actual measurable quantity. On the other hand, $\mathbf{H}$ are ancilla variables, that are used to model dependencies between the observed data. We also refer to $\mathbf{V}$ and $\mathbf{H}$ as visible or hidden variables respectively. A model that is less general than the BM, but easier to handle is the *Deep Boltzmann Machines* (DBM). The DBM is a BM, where the nodes associated to visible variables are not connected. Strictly speaking this means that the distribution $P(\mathbf{V})$, defined by a DBM, does factorize as

$$P(\mathbf{V}) = \prod_{V \in \mathbf{V}} P(V) \ . \tag{7.2}$$

The same holds for the hidden variables, but we have the freedom to introduce additional sets of hidden variables $\mathbf{H}^{(i)}$ that fulfill the properties [57]

$$P(\mathbf{H}^{(i)}) = \prod_{H^{(i)} \in \mathbf{H}^{(i)}} P(H^{(i)}), \ P(\mathbf{H}^{(i)}|\mathbf{H}^{(i-1)}) = \prod_{H^{(i)} \in \mathbf{H}^{(i)}} P(H^{(i)}|\mathbf{H}^{(i-1)}) \ . \tag{7.3}$$

Thus, consecutive hidden variables are conditional dependent, but hidden variables from the same set are independent. At his point we want to establish a common terminology between DBMs and ANNs, such as emphasize their connections.



visible **V**
v_1 v_2 v_3 ... $v_{n^{(0)}}$
a_1 a_2 a_3 ... $a_{n^{(0)}}$
$W^{(1)}$
1st hidden **H**$^{(1)}$
$c^{(1)}_1$ $c^{(1)}_2$ $c^{(1)}_3$ $c^{(1)}_4$ ... $c^{(2)}_{n^{(1)}}$
$W^{(2)}$
$c^{(2)}_1$ $c^{(2)}_2$ $c^{(2)}_3$ ... $c^{(2)}_{n^{(2)}}$
2nd hidden **H**$^{(2)}$
$h^{(2)}_1$ $h^{(2)}_2$ $h^{(2)}_3$ ... $h^{(2)}_{n^{(2)}}$

Figure 7.2: Here we have an example for a Deep Boltzmann Machines with two layers of hidden variables.

In terms of the ANN we group neurons, that are not interconnected and of the same activation function, to one layer. In close relation we refer to all visible variables as the *visible layer*. This specific layer of the DBM resembles the input layer of an ANN since it takes the observed data. When speaking about ANNs all layer after the input layer, are referred to as hidden layers. The same term is used for DBMs, where variables from a certain set of hidden variables form together a layer. Thus, we associate to every hidden layer, like in Fig. 7.2, a set of hidden variables $\mathbf{H}^{(i)}$. Fig. 7.2 is an example for a DBM with two hidden layers. The number of variables in every layer is denoted with $n^{(i)}$, in the special case of the visible layer $i = 0$. The name *depth*, which describes the number of layer $\mathcal{L}$ is adopted directly from ANNs to DBMs. The most obvious difference between a DBM and an ANN is that the DBM has no output. Like we establish in Sec. 3.1 a DBM does not output a certain probability $P(\mathbf{V} = \mathbf{v}) = p(\mathbf{v})$, like an ANN, but is the probability distribution itself. To express this probability distribution, we exploit the way a DBM factorizes over a graph like in Fig. 7.2. One can see what we mean by this when taking the example of Fig. 7.2

$$P(\mathbf{V} = \mathbf{v}, \mathbf{H}^{(1)} = \mathbf{h}^{(1)}, \mathbf{H}^{(2)} = \mathbf{h}^{(2)}) = p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}) = p(\mathbf{v}, \mathbf{h}^{(1)})p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) =$$
$$\mathrm{e}^{\log(p(\mathbf{v},\mathbf{h}^{(1)}))+\log(p(\mathbf{h}^{(1)},\mathbf{h}^{(2)}))} = \frac{1}{Z}\,\mathrm{e}^{-E(\mathbf{v},\mathbf{h}^{(1)},\mathbf{h}^{(2)})} \,,$$

where we exploit the positivity of probability distributions to express the joint probability of a DBM with an *energy function* E. The energy function E can be generalized for an arbitrary number of hidden layers and takes the form

$$E(\mathbf{v}, \{\mathbf{h}^{(i)}\}_{i=1}^{\mathcal{L}}) = \mathbf{v}^T\mathbf{W}^{(1)}\mathbf{h}^{(1)} + \sum_{i=1}^{\mathcal{L}} (\mathbf{h}^{(i)})^T\mathbf{W}^{(i+1)}\mathbf{h}^{(i+1)} + \mathbf{a}^T \cdot \mathbf{v} + \sum_{i=1}^{\mathcal{L}} \mathbf{c}^{(i)} \cdot \mathbf{h}^{(i)} \,. \quad (7.4)$$

In Eq. (7.4) conditional dependencies are expressed by the *weights* $\{\mathbf{W}^{(i)}\}$, which establish an interaction between variables of consecutive layer. The variables $\mathbf{a}$ and $\{\mathbf{c}^{(i)}\}$ are called *biases* and are not-interacting weights associated to every node. Like in Sec. 4 we denote all parameters, that specify the model by $\theta = \{\{\mathbf{W}^{(i)}\}, \{\mathbf{c}^{(i)}\}, \mathbf{a}\}$. The partition function $Z(\theta)$ is thus defined as

$$Z(\theta) = \sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\{\mathbf{h}^{(i)}\})} , \tag{7.5}$$

which leads to the distribution of the visible variables [4]

$$p(\mathbf{v}) = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\{\mathbf{h}^{(i)}\})} . \tag{7.6}$$

In equation Eq. (7.6) the summation runs over $\mathbf{h} \in \{-1, 1\}^{n^{(1)}+n^{(2)}+\ldots+n^{(\mathcal{L})}}$ which results in an exponential sum that cannot be brought in a analytic form. Due to this, Eq. (7.6) is intractable and can in practice only carried out by employing approximations [22, 4]. The only case, in which this problem does not occur is $\mathcal{L} = 1$. This special case of a single hidden layer is referred to as *Restricted Boltzmann Machines* (RBM) and has the following energy function

$$E(\mathbf{v}, \mathbf{h}) = \mathbf{v}^T W \mathbf{h} + \mathbf{a} \cdot \mathbf{v} + \mathbf{c} \cdot \mathbf{h} \tag{7.7}$$

When we plug this energy function into Eq. (7.6) we obtain this analytic form for the RBM [16]

$$p(\mathbf{v}) \propto \mathrm{e}^{\mathbf{a}\cdot\mathbf{v}} \prod_{i}^{n^{(1)}} 2\cosh\left(c_i + \sum_{j}^{n^{(0)}} W_{ij} h_i v_j \; . \right) \tag{7.8}$$

With Eq. (7.8) we obtain a useful form to parameterize a probability distribution, and by a distribution we mean *any* distribution. It can be shown that RBMs are capable of approximating any discrete probability distribution, over a random vector $\mathbf{v} \in \{-1, 1\}^{n^{(0)}}$, arbitrarily well by increasing $n^{(1)}$ [21]. This is a very similar statement to the universality statement regarding ANNs, but opposite to that case we have here only one unknown parameter. Thus, it might appear easy to find the right model and train it to approximate a desired distribution. But as it turns out training RBMs is not straight forward and other not so obvious difficulties appear, which we discuss in the following part of this section.

7.2. Properties of the Restricted Boltzmann Machines

As we bring up in Sec. 3.2 the RBM, as it is introduced in Eq. (7.8), can be used as variational ansatz to infer the ground-state of many-particle Hamiltonians like the ANNs in Sec. 4.4. An advantage of the RBM as ansatz is obviously the little number of architectures one can create. Actually, there is only one, with its only free variable to set, being the size of the hidden layer. If one has set the size of the hidden layer, the next step is to determine the parameters θ that minimize the variational energy. Like the gradient descent algorithm with back-propagation for ANNs (Sec. 4.4), there are some specific training algorithms for RBMs [57]. Their discussion exceeds the scope of this work since the reason why we do not use RBMs as variational ansatz does not even arise from problems of the optimization algorithm.

One first clear drawback of the RBM is that it does not allow for negative values Eq. (7.8), which is in fact reasonable for a probability distribution, but a constraint in terms of quantum wave-functions. This is barley an inconvenience as we can allow the parameters θ to be complex, such that the RBM can take complex values [21]. As it turns out, this does not resolve the main restraint of the RBM. In a rigorous proof it can be shown that Deep Boltzmann Machines (as introduced in Sec. 7.1) can efficiently represent the ground-state of any *k-local* many-particle Hamiltonians with a *polynomial-size gap*. Here *k-local* means that the interaction term of the regarding Hamiltonian is restricted to couple k particles. A *polynomial-sized gap* refers to the energy difference, between the ground-state and the first excited state of a Hamiltonian, decreasing with $1/\text{poly}(N)$, where N is the number of particles in the system. Unfortunately, we know that the DBM is not tractable in an actual optimization attempt. Thus, we look at the RBM, but it turns out that the RBM can *not* efficiently represent the ground-states that a DBM can. In fact, the RBM cannot even efficiently *approximate* the same states that the DBM can represent efficiently and exactly [58]. We want to stress here again, that *efficiently represent* means that the number of parameters required to express a certain state is growing at most polynomial with the system size $\text{poly}(N)$. Thus, the RBM is not as practicable as one would desire. Nevertheless, investigations treating the Restricted Boltzmann Machines uncovered its direct relation to Tensor-Networks. Tensor Networks are a common and very successful variational approach to find the ground-state of quantum many-body Hamiltonians. The term *Tensor Network* sums up a set of different approaches that are conceptually similar. In particular direct relations between the RBM and Matrix Product States, Entangled Plaquette States and String Bond States were found. This has also led to the discovery of exact representations of states like the *Toric Code State* or *Graph States* using an RBM [22, 21]. But this is the RBM from the Tensor-Network point of view and our discussion focuses on *Machine Learning* the ground-state quantum wave-function. Hence, we introduce a different use for the RBM form the point of view of Machine Learning.

7.3. Layer-Wise Unsupervised Pre-training

As we mention in Sec. 4.4 the training of Artificial Neural Networks is a somewhat cumbersome procedure. One of the reasons for this is that the *ideal* starting weights are unknown, and the weights have to be initialized somehow randomly. In all the previous tests we do this with a random Gaussian distribution, but techniques were developed that are a more reliable initialization method. The method we look into here is called *Layer-Wise Unsupervised Pre-training* and is implemented using the Restricted Boltzmann Machines. Having a data-set, in this case $\mathcal{T} = \{-1, 1\}^N$, we can calculate how probable it is to find all this data, given an RBM with certain parameters θ

$$\mathcal{L}(\mathcal{T}, \theta) = p(\mathcal{T}|\theta) = \prod_{\mathbf{v}\in\mathcal{T}} p(\mathbf{v}|\theta) \ . \tag{7.9}$$

The probability $\mathcal{L}(\mathcal{T}, \theta)$ is the so called *likelihood*. Maximizing the likelihood means that we search parameters θ for which the data $\mathcal{T}$ occur with the highest possible probability. Maximizing the likelihood is equivalent to maximizing the *log-likelihood* [21]

$$\log \mathcal{L}(\mathcal{T}, \theta) = \sum_{\mathbf{v}\in\mathcal{T}} \log p(\mathbf{v}|\theta) \ . \tag{7.10}$$

Consider now the TDT network from Fig. 5.31 with homogeneous hidden layer-size M. It is a three-layer network, where the last layer is naturally the output layer. Every layer of this network can be interpreted as a independent RBM. In this picture the output of the foregoing feed forward layer is the input of the visible units of the subsequent RBM. The point of all this is that we can initialize an RBM, with the energy function

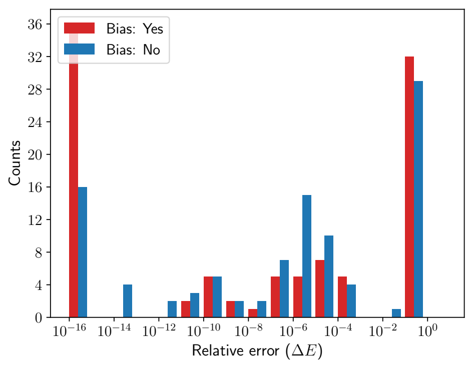
$$E(\mathbf{v}, \mathbf{h}) = \mathbf{v}^T \mathbf{W} \mathbf{h} \ , \tag{7.11}$$

that has N visible units and M hidden units. Then we maximize the log-likelihood for the configuration set $\mathcal{T}$, the obtained weight matrix W is then used to initialize the first layer in the TDT network. In a similar way we can initialize every layer of an ANN. This pre-training procedure offers the advantage that every layer is able to extract important features, from the data that it is fed to. Thus, when we begin the actual training we can start out closer to the minimum and achieve it with higher reliability. To clarify all this, we state now the pre-training procedure for an arbitrary network [59, 54]:

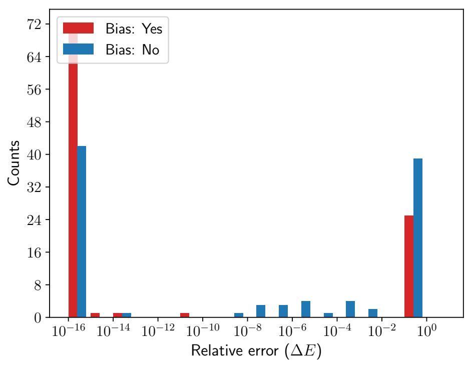
1. Initialize an RBM with the energy function $(\mathbf{W}, \mathbf{c}) \in \mathbb{R}^{N\times n^{(1)}+n^{(1)}}$, $E(\mathbf{v}, \mathbf{h}) = \mathbf{v}^T \mathbf{W} \mathbf{h} + \mathbf{c} \cdot \mathbf{h}$ and maximize $\log \mathcal{L}(\mathcal{T}, \theta)$ using a greedy algorithm (we use the *BFGS* algorithm [60]).

2. Set the ANN parameters to $\mathbf{W}^{(1)} = \mathbf{W}$ and $\mathbf{b}^{(1)} = \mathbf{c}$.
3. Set $\mathcal{T}^{(0)} = \mathcal{T}$, $\mathbf{a}^{(0)} = \mathbf{v}$ and repeat the following for all hidden layers $0 \leq i < \mathcal{L} - 1$:
    a) Calculate $\mathbf{a}^{(i+1)} = \mathbf{f}^{(i+1)}(\mathbf{W}^{(i+1)}\mathbf{a}^{(i)} + \mathbf{b}^{(i+1)})$ for all $\mathbf{a}^{(i)} \in \mathcal{T}^{(i)}$, which yields $\mathcal{T}^{(i+1)} = \{\mathbf{a}^{(i+1)}\}$.
    b) Initialize an RBM with the energy function $(\mathbf{W}, \mathbf{c}) \in \mathbb{R}^{n^{(i+1)} \times n^{(i+2)} + n^{(i+2)}}$, $E(\mathbf{a}^{(i+1)}, \mathbf{h}) = (\mathbf{a}^{(i+1)})^T \mathbf{W}\mathbf{h} + \mathbf{c} \cdot \mathbf{h}$ and maximize $\log \mathcal{L}(\mathcal{T}^{(i+1)}, \theta)$ using a greedy algorithm.
    c) Set the ANN parameters to $\mathbf{W}^{(i+2)} = \mathbf{W}$ and $\mathbf{b}^{(i+1)} = \mathbf{c}$.

We apply the pre-training algorithm to the case of the TDT (with homogeneous layer-size $M = 9$ and $M = 10$) network with the aim to find the ground-state of Hamiltonian $\mathcal{H}_{ms}$ $N = 8$. The results can be seen in Fig. 7.3a.



(a) Model: TDT with $M = 9$, Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 8$



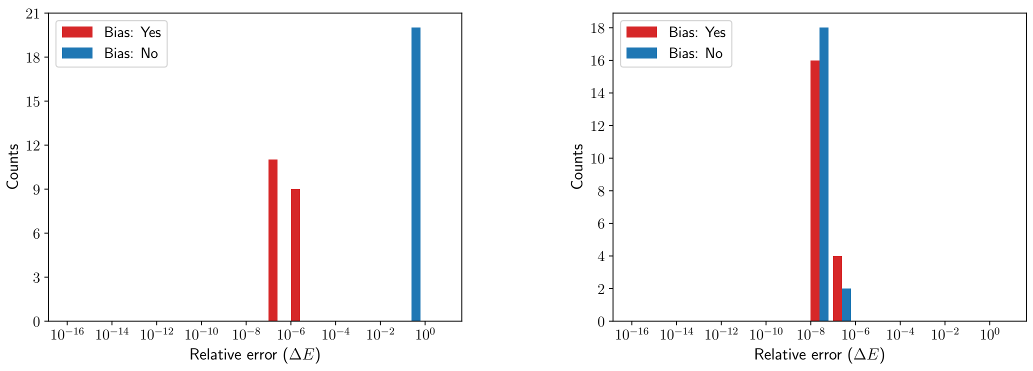
(b) Model: TDT with $M = 10$, Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 8$

Figure 7.3: Number of optimization attempts: 100; We treat Hamiltonian $\mathcal{H}_{ms}$ $N = 8$ with the TDT model and compare the results of randomly initialized weights to pre-trained weights. In Fig. 7.3a we have a homogeneous layer of size of $M = 9$, and in Fig. 7.3b a layer-size of $M = 10$.

For comparison to the pre-trained networks in Fig. 7.3a and Fig. 7.3b we have the results for the same networks, but without pre-training also at display in this figure. It turns out that the reliability to obtain machine precision approximately doubles

$$M = 9 : 0.16 \ \rightarrow \ 0.36$$
$$M = 10 : 0.42 \ \rightarrow \ 0.72$$

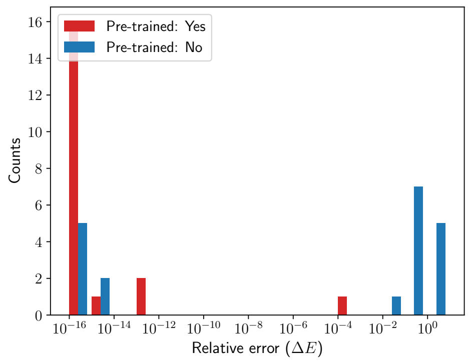
using pre-training in this case. This technique might be very useful, when one has to employ Monte Carlo sampling to treat larger systems. The pre-training is sufficiently simple and yields a good starting point for the simulation, as Fig. 7.3a shows. The only uncontrolled part, that is left in the initialization, is that the Boltzmann Machines does not always find the same maximum, which might be one cause for the unreliability in Fig. 7.3a. The success of this method motivates us to revisit some problems from Sec. 5, where we pre-train the networks this time. First, we look at Hamiltonian $\mathcal{H}_{ps}$ as in Sec. 5.2 for the Models TTTT and SRS in Fig. 7.4.



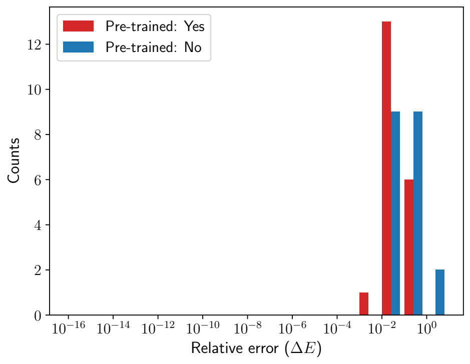
(a) Model: TTTT **(b)** Model: SRS

Figure 7.4: Hamiltonian: $\mathcal{H}_{ps}$, System size: $N = 8$, Pre-trained: Yes

In Fig. 7.4a one cannot really see an improvement over Fig. 5.5a. For the SRS we see in Fig. 7.4b that we still get the same maximal precision as in Fig. 5.9a, but with higher probability. Thus, there is no qualitative difference to the case without pre-training and only a small quantitative difference. We get a different picture when we look at Hamiltonian $\mathcal{H}_{ms}$. We do so for $N = 10$ and the restricted training set $\mathcal{T}_0$ with the TTT network (without bias) and the TRT network (with bias). The result is shown in Fig. 7.5



(a) Model: TTT



(b) Model: TRT

Figure 7.5: Hamiltonian: $\mathcal{H}_{ms}$, System size: $N = 10$, Training space: $\mathcal{T}_0$

Comparing Fig. 7.5b to the results from Fig. 5.21b, where we also treat the anti-symmetric problem with a TRT network, we find that pre-training does not help us to overcome the precision bound we find also without pre-training. In Sec. 5.3 we find that TTT yields machine precision, but only with a small probability (Fig. 5.18b). In Fig. 7.5a we find that pre-training improves heavily the training success, instead of 5 out of 100 times it yields 16 out of 20 times machine precision. The conclusion we draw from the comparison of pre-trained networks to randomly initialized networks is that pre-training does not improve the ability to approximate the wave-function. This can best be seen at the example of TTTT (Fig. 7.4a), where pre-training does not help to obtain higher precision than $\Delta E \approx 10^{-8}$. In general, it allows to increase the training success of a promising ansatz.

8. Discussion

In this work we investigated how changing the degrees of freedom, which we have by design of the concept of Artificial Neural Networks, influences the capability of finding the ground-state of the anti-ferromagnetic Heisenberg Hamiltonian on a small even numbered spin-ring. The concept of ANNs is formed of three ingredients: the neural network itself, the cost function and the training procedure. While we tried to cover the essentials about the cost function and training (see Sec. 3 and Sec. 4.4), the center of our investigation was the neural network itself. An ANN, as a function, is defined by the number of layers, each layer's activation function and number of neurons per layer. At the beginning of our discussion we examined the impact of depth on certain combinations of activation functions, which means that we analyzed the interplay of the first two of the just stated points. What we found is that increasing the depth of certain networks enlarges the accessible function space, while keeping a constant total number of network parameters. At a later point we were able to observe that a network with three layers (TDT) can be more parameter-efficient than a network with two layers (DT). Even though both networks rely on exploiting redundancy (see Sec. 5.5.3) and spin-inversion symmetry (see Sec. 5.4) the network with more layers requires less parameters to yield a computationally exact ground-state energy. So, deeper networks access a greater function space and are more parameter-efficient, though in Sec. 5.6 we cast our heuristic findings into guidelines on how to construct a network and we recommend using at most a four-layer network. This is due to the limitations of the used optimization algorithm (see Sec. 4.3 and Sec. 4.4). The optimization algorithm is a gradient-based algorithm, which has some problems by itself and has more problems when back-propagation is used to calculate the gradient [53, 4]. We were able to observe the typical problems of the back-propagation algorithm (see Sec. 5.2) in our test data and found a way to make moderately deep networks feasible by employing activation functions with strong gradients. Though, this strategy worked not for very deep networks, which is the reason for our recommendation regarding depth. Note that we use Artificial Neural Networks only as function approximates, but the formalism of ANNs allows for much more interesting training and application strategies. In the case of *Generative Adversarial Networks* [61] one network generates samples while another network categorizes real (not generated) and generated samples. The output of the networks is combined in a common cost-function, which allows for simultaneous training. A similar approach could lead to a network generating configuration vectors and another network acting as variational wave-function. Even when this or other strategies (e.g. Deep Residual Learning [42]) lead to feasible deep networks it is not clear how broad the spectrum of application for a single network architecture is (there is no such thing as free lunch [62]).

The findings that allowed us to design TDT, like that redundant neurons and functions with strong gradients can increase the success of training, or that increasing depth helps to decrease the required number of parameters are represented in the rules form Sec. 5.6. Those rules reflect that there is no clear *one suits all* solution, which is also consistent with what others found. For example, Saito *et al.* [18] found for their combination of activation function and treated Hamiltonian, that the fully connected network works best with one hidden layer. In contrast to that they had the best results for convolutional networks with two convolution layers. Another result found by Cai *et al.* [17] is that for his application a four-layer fully connected network is working best. What these results and the ones from Choo *et al.* [19] have in common is that they differ significantly from the results provided by well-established methods, like Density Matrix Renormalization

Group (DMRG) [2] or Exact Diagonalization (e.g. Choo *et al.* mention a relative error of $10^{-5} - 10^{-3}$ in terms of eigenstate energies, provided by DMRG).

The Matrix Product State (MPS) ansatz [3] is known to provide very reliable results for one-dimensional systems of gaped Hamiltonians. The entanglement entropy of ground-states of Hamiltonians with gaps and short ranged interactions in a single dimension, is not extensive. This is the so-called *area-law* which states that the entanglement between two parties (A of size $M^d < N^d$ and B of size $N^d - M^d$) in a ground-state of a just described Hamiltonian behaves like $S(A|B) \propto M^{d-1}$, where d is the lattice dimension. The working principle of MPS is to exploit the constant entanglement entropy in one-dimension. The MPS ansatz provides that no matter how the one dimensional state is bi-partitioned, it always yields the same amount of entanglement entropy. One can regulate this amount with the *bond dimension* D. The bond dimension allows to control the precision of the approximation, hence for D being large enough the approximation becomes exact. The ANN ansatz lacks such a control parameter. Of course, we can increase the node number but this does not consistently increase the precision of the approximation. From Fig. 5.37 we learn that for a machine precise energy, the scaling of the number of parameters is exponential, but there is not much in between a bad approximation and the maximal precise approximation. It either works or it does not. Maybe the scaling can be improved, when one settles for a fixed second layer-size (which is the one that causes the quadratic parameter growth) and only increases the first layer. But still this holds then for this specific Hamiltonian and architecture. Though the advantage of ANNs can be seen at the point when MPS breaks down. In two dimensions entanglement is way more complicated than in one dimension, and MPS cannot capture this (a more suitable approach would be *Projected Entangled Pair States* [3]), as it is designed for non-extensive entanglement entropy. But ANNs do not rely on such things, their approximation capability is connected to the complexity of the actual wave-function components the network has to learn. Thus, there is nothing that prevents ANNs from working in all dimensions the same as they do in one dimension.

Despite the difference between Artificial Neural Networks and Tensor Networks (TN, which is the overall terms to which MPS belongs) both fields can profit from each other. Relations between neural networks and Tensor Networks might lead to deeper theoretical understanding of deep neural networks and perhaps explain their *unreasonable success* [63, 64]. In the field of neural networks, the reuse of information, e.g. information that was already shown to the network and is later brought up again, is something well established. This technique is not yet well established in the field of TNs, but it allows for example to account for logarithmic corrections to the area-law in one dimension [20]. The relation between efficient representation of physical systems and highly complex discriminative or generative tasks is a field of study that is quite young but holds rich promises in terms of better algorithms and deeper understanding of information processing.

9. Acknowledgment

I would like to thank Univ. Prof. Dr. Andreas Läuchli for the constant discussions and his helpful advises. I especially want to express my gratitude to Thomas Lang for his continuous support and for always having a spare second. I would also like to thank Michael Rader for sharing some of his expertise on high performance computing and on other useful things. Finally, I would like to thank Lei Wang for the helpful discussion and paying for lunch.

Appendices

A. Expressiveness and Reconstruction Theorems

Consider a neural network with two layers, then the set

$$\mathcal{R}_k^n = \left\{ h : \mathbb{R}^k \to \mathbb{R} | h(\mathbf{v}) = \sum_i^n w_i^{(1)} f^{(1)}(\mathbf{w}_i^{(1)} \cdot \mathbf{v} + b_i^{(1)}) \right\} \tag{A.1}$$

is the set that contains all functions that can be expressed with such a network. The set of all functions that can be implemented by a two-layer network is given as

$$\mathcal{R}_k = \bigcup_{n=1}^{\infty} \mathcal{R}_k^n. \tag{A.2}$$

Consider the function space $\mathcal{F} = \mathcal{L}^p(\mu)$ or $\mathcal{F} = \mathcal{C}(\mu)$. A subset $S \subset \mathcal{F}$ is said to be dense in $\mathcal{F}$ when for arbitrary $f \in \mathcal{F}$ and $\epsilon > 0$, there is $g \in S$ such that

$$\rho_{p,\mu}(f, g) < \epsilon. \tag{A.3}$$

The distance measure $\rho_{p,\mu}$ is defined by the $\mathcal{L}^p$ norm of the difference between f and g. Given these definitions we can introduce the theorems that lead to the statement about approximation capabilities in Sec. 4.2.

Theorem A.1. *If $f^{(1)}$ is unbounded and non-constant, then $\mathcal{R}_k$ is dense in $\mathcal{L}^p(\mu)$ for all finite measure μ on $\mathbb{R}^k$.*

Theorem A.2. *If $f^{(1)}$ is continuous, bounded and non-constant, then $\mathcal{R}_k$ is dense in $\mathcal{C}(\mu)$ for all compact subsets X of $\mathbb{R}^k$.*

For additional details see Hornik *et al.* [15].

As we further mention in Sec. 4.2, in order to reconstruct a function from a *Ridgelet transform* the kernel functions of the transformation and reconstruction need to be *admissable*. For this definition we need the space of *rapidly decreasing functions* $\mathcal{S}(\mathbb{R})$ and the space of *tempered distributions* $\mathcal{S}'(\mathbb{R})$. Further we denote the Fourier transform with $\mathcal{F} : f \mapsto \hat{f}$.

Definition A.1. A pair $(f, g) \in \mathcal{S}(\mathbb{R}) \times \mathcal{S}'(\mathbb{R})$ is said to be admissible when there exists a neighborhood $\Omega \subset \mathbb{R}$ of 0 such that $\hat{\eta} \in \mathcal{L}^1_{loc}(\Omega \setminus \{0\})$, and the integral

$$K_{f,g} = (2\pi)^{m-1} \left(\int_{\Omega \setminus \{0\}} + \int_{\mathbb{R} \setminus \Omega} \right) \frac{\overline{\hat{f}(\zeta)} \hat{\eta}(\zeta)}{|\zeta|^m} \, \mathrm{d}\zeta, \;\; m \in \mathbb{N} \tag{A.4}$$

converges and is not zero, where $\int_{\Omega \setminus \{0\}}$ and $\int_{\mathbb{R} \setminus \Omega}$ are understood as Lebesgue's integral and the action of $\hat{\eta}$, respectively.

Finally, we can state the conditions under which a reconstruction is possible, note that the integer m from Eq. (A.4) is the same as in the next definition.

Definition A.2. Let $q \in \mathcal{L}^2(\mathbb{R}^m)$ and $(f, g) \in \mathcal{S}(\mathbb{R}) \times \mathcal{S}'(\mathbb{R})$ be admissibly decomposable with $K_{f,g} = 1$. Then,

$$\mathcal{R}_f \mathcal{R}^\dagger{}_g q \to q, \text{ in } \mathcal{L}^2 \,. \tag{A.5}$$

For further details we refer to Sonodo *et al.* [49].

To introduce the theorem regarding the advantage of a three-layer network over a two-layer network we need to introduce two lemmas first [50].

Lemma A.3. *Given the activation function σ, there is a constant $c_\sigma \geq 1$ (depending only on σ) such that the following holds: For any L-Lipschitz function $f : \mathbb{R} \to \mathbb{R}$ which is constant outside a bounded interval $[-R, R]$, and for any δ, there exist scalars $a, \{\alpha_i, \beta_i, \gamma_i\}_{i=1}^{w}$ where $w \leq c_\sigma \frac{RL}{\delta}$, such that the function*

$$h(x) = a + \sum_{i=1}^{w} \alpha_i \sigma(\beta_i x - \gamma_i) \tag{A.6}$$

satisfies

$$\sup_{x \in \mathbb{R}} |f(x) - h(x)| \leq \delta \,. \tag{A.7}$$

Lemma A.4. *The activation function σ is (Lesbegue) measurable and satisfies*

$$|\sigma(x)| \leq C(1 + |x|^\alpha) \tag{A.8}$$

for all $x \in \mathbb{R}$ and for some constants $C, \alpha > 0$.

This leads to the theorem of the advantage of a three-layer network over a two-layer network.

Theorem A.5. *Suppose the activation function $\sigma(\cdot)$ satisfies Lemma A.3 with constant c_σ, as well as Lemma A.4. Then there exists universal constants $c, C > 0$ such that the following holds: For every dimension $d > C$, there is a probability measure μ on $\mathbb{R}^d$ and a function $g : \mathbb{R}^d \to \mathbb{R}$ with following properties:*

- *g is bounded in $[-2, +2]$, supported on $\{\mathbf{x} : |\mathbf{x}| \leq C\sqrt{d}\}$, and expressible by a three-layer network of width $Cc_\sigma d^{19/4}$.*
- *Every function f, expressed by a two-layer network of width at most ce^{cd}, satisfies*

$$\mathbb{E}_{x\sim\mu}[f(\mathbf{x}) - g(\mathbf{x})]^2 \geq c. \tag{A.9}$$

B. Derivatives

In this part of the appendix we provide the derivatives for the used cost function and the sparse layer, such as the convolution layer. We state the derivatives in terms of the back-propagation algorithm:

- Cost function for when $f(z^{(\mathcal{L})}) \in \mathbb{R}$:

$$\delta_j^{(\mathcal{L})} = \frac{\partial E(W,b;\{\mathbf{v}\})}{\partial z_j^{(\mathcal{L})}} \cdot f'(z_j^{(\mathcal{L})}) =$$

$$\left(\frac{\sum_l f(z_l^{(\mathcal{L})})^2 \left(2\mathcal{H}_{jj} f(z_j^{(\mathcal{L})}) + 2\sum_{l\neq j} \Re(\mathcal{H}_{jl}) f(z_l^{(\mathcal{L})})\right)}{\sum_l f(z_l^{(\mathcal{L})})} - ... \right.$$

$$\left. ... - \frac{2f(z_j^{(\mathcal{L})}) \left(\sum_{lg} \mathcal{H}_{lg} f(z_l^{(\mathcal{L})}) f(z_g^{(\mathcal{L})})\right)}{\sum_l f(z_l^{(\mathcal{L})})} \right) f'(z_j^{(\mathcal{L})})$$

- Sparse linear operation:

$$\nabla_{W_{jg}^{(i)}} E(W,b;\{\mathbf{v}\}) = \delta_j^{(i+1)} a^{(i)}_{(j+g(\mathrm{lrf}-\mathrm{ol}))\bmod n^{(i)}}$$

- Convolution operation:

$$\nabla_{W_{jg}^{(i)}} E(W,b;\{\mathbf{v}\}) = \delta_j^{(i+1)} \sum_{l=1}^{n^{(i)}} a_{l+g}^{(i)}$$

C. Homogeneous Layer-Sizes

Throughout this work we use histograms to show the relation between the precision a neural network can achieve and how often it succeeds in doing so. These histograms are, besides some highlighted special cases, always created with a network for which it holds that

$$|\theta| \approx \mathcal{D} \ , \tag{C.1}$$

this means that the total number of network parameters is nearly equal to the dimension of the Hilbert space, but never more. From this we can calculate the number of neurons in all layers, besides in the input and output layer, under the assumption that the hidden layers are of equal size

$$n^{(0)} = N$$
$$n^{(\mathcal{L})} = 1$$
$$n^{(1)} = n^{(2)} = ... = n^{(\mathcal{L}-1)} = M \ .$$

The layer-size M does of course also depend on whether we use bias or not, this is also reflected in Eq. (5.5), which is the equation we use to calculate M. One can find the results for system sizes $N = 8$ and $N = 10$ for $2 \leq \mathcal{L} \leq 12$ in Tab. 5

Depth ($\mathcal{L}$)	$N = 8$, Bias: Yes	$N = 8$, Bias: No	$N = 10$, Bias: Yes	$N = 10$, Bias: No
2	25	28	85	93
3	11	12	26	26
4	8	9	19	20
5	7	7	16	16
6	6	6	14	14
7	5	6	12	13
8	5	5	11	12
9	4	5	10	11
10	4	5	10	10
11	4	4	9	10
12	4	4	9	9

Table 5: This table contains the homogeneous layer-size M for systems sizes $N = 8$ and $N = 10$.

References

[1] F. Verstraete, V. Murg, and J. I. Cirac, "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems," *Advances in Physics*, vol. 57, pp. 143–224, Mar 2008. [Online]. Available: http://adsabs.harvard.edu/abs/2008AdPhy..57..143V

[2] U. Schollwöck, "The density-matrix renormalization group in the age of matrix product states," *Annals of Physics*, vol. 326, no. 1, pp. 96 – 192, Jan 2011, january 2011 Special Issue. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003491610001752

[3] N. Schuch, "Condensed matter applications of entanglement theory," *arXiv:1306.5551*, 2013. [Online]. Available: https://arxiv.org/abs/1306.5551

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2017.

[5] L. Deng and D. Yu, "Deep learning: Methods and applications," Tech. Rep., May 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[7] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature Communications*, vol. 5, Jul 2014, article. [Online]. Available: http://dx.doi.org/10.1038/ncomms5308

[8] V. Mitra and H. Franco, "Coping with Unseen Data Conditions: Investigating Neural Net Architectures, Robust Features, and Information Fusion for Robust Speech Recognition," *Proc.INTERSPEECH 2016*, pp. 3783–3787, Sep 2016.

[9] A. Singhal, P. Sinha, and R. Pant, "Use of deep learning in modern recommendation system: A summary of recent works," *International Journal of Computer Applications*, vol. 180, no. 7, pp. 17–22, Dec 2017. [Online]. Available: http://www.ijcaonline.org/archives/volume180/number7/28811-2017916055

[10] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on.* IEEE, 2017, pp. 1–12.

[11] "Nvidia volta ai architecture," Jul 2018. [Online]. Available: https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015, pp. 1–9.

[13] R. Raul, *Neural networks: a systematic introduction.* Springer, 1996.

[14] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, May 2009.

[15] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190009T

[16] G. Carleo and M. Troyer, "Solving the quantum many-body problem with artificial neural networks," *Science*, vol. 355, no. 6325, pp. 602–606, 2017. [Online]. Available: http://science.sciencemag.org/content/355/6325/602

[17] Z. Cai and J. Liu, “Approximating quantum many-body wave-functions using artificial neural networks,” *arXiv:1704.05148*, 2017. [Online]. Available: https://arxiv.org/abs/1704.05148

[18] H. Saito and M. Kato, “Machine learning technique to find quantum many-body ground states of bosons on a lattice,” *Journal of the Physical Society of Japan*, vol. 87, no. 1, p. 014001, 2018. [Online]. Available: https://doi.org/10.7566/JPSJ.87.014001

[19] K. Choo, G. Carleo, N. Regnault, and T. Neupert, “Symmetries and many-body excited states with neural-network quantum states,” *arXiv:1807.03325*, 2018. [Online]. Available: https://arxiv.org/abs/arXiv:1807.03325

[20] Y. Levine, O. Sharir, N. Cohen, and A. Shashua, “Bridging many-body quantum physics and deep learning via tensor networks,” *arXiv:1803.09780*, 2018. [Online]. Available: https://arxiv.org/abs/1803.09780

[21] S. R. Clark, “Unifying neural-network quantum states and correlator product states via tensor networks,” *Journal of Physics A: Mathematical and Theoretical*, vol. 51, no. 13, p. 135301, 2018. [Online]. Available: http://stacks.iop.org/1751-8121/51/i=13/a=135301

[22] I. Glasser, N. Pancotti, M. August, I. D. Rodriguez, and J. I. Cirac, “Neural-network quantum states, string-bond states, and chiral topological states,” *Phys. Rev. X*, vol. 8, p. 011006, Jan 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.8.011006

[23] J. Chen, S. Cheng, H. Xie, L. Wang, and T. Xiang, “Equivalence of restricted boltzmann machines and tensor network states,” *Phys. Rev. B*, vol. 97, p. 085104, Feb 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.97.085104

[24] “Cs231n convolutional neural networks for visual recognition,” Jul 2018. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[25] G. D. Mahan, *Condensed matter in a nutshell.* Princeton University Press, 2011.

[26] R. C. Ashoori, “Electrons in artificial atoms,” *Nature*, pp. 379–413, Feb 1996. [Online]. Available: http://dx.doi.org/10.1038/379413a0

[27] J. G. Bednorz and K. A. Müller, “Possible high t_c superconductivity in the ba-la-cu-o system,” *Zeitschrift fur Physik B Condensed Matter*, vol. 64, pp. 189–193, Jun 1986, provided by the SAO/NASA Astrophysics Data System. [Online]. Available: http://adsabs.harvard.edu/abs/1986ZPhyB..64..189B

[28] U. Schollwöck, J. Richter, D. J. Farnell, and R. F. Bishop, *Quantum magnetism.* Springer, 2004.

[29] R. Coldea, D. A Tennant, E. M Wheeler, E. Wawrzynska, D. Prabhakaran, M. Telling, K. Habicht, P. Smeibidl, and K. Kiefer, "Quantum criticality in an ising chain: Experimental evidence for emergent e8 symmetry," vol. 327, pp. 177–180, Jan 2010.

[30] L. Henkens, K. Diederix, T. Klaassen, and N. Poulis, "Study of heisenberg linear chains in cubef4.5h2o," *Physica B+C*, vol. 81, no. 2, pp. 259 –274, 1976. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0378436376900619

[31] A. Auerbach, *Interacting electrons and quantum magnetism.* Springer, 2012.

[32] G. B. Arfken, *Mathematical methods for physicists.* Academic Press, 1985.

[33] W. Heisenberg, "Zur theorie des ferromagnetismus," *Zeitschrift für Physik*, vol. 49, no. 9, pp. 619–636, 1929.

[34] M. Karbach and G. Muller, "Introduction to the bethe ansatz i," *arXiv:cond-mat/9809162*, 1998. [Online]. Available: https://arxiv.org/abs/cond-mat/9809162

[35] K. Bärwinkel, H.-J. Schmidt, and J. Schnack, "Ground-state properties of antiferromagnetic heisenberg spin rings," *Journal of Magnetism and Magnetic Materials*, vol. 220, no. 2, pp. 227–234, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304885300004819

[36] "Antiferromagnetism," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 232, no. 1188, pp. 48–68, 1955. [Online]. Available: http://rspa.royalsocietypublishing.org/content/232/1188/48

[37] P. Henelius and A. W. Sandvik, "Sign problem in monte carlo simulations of frustrated quantum spin systems," *Phys. Rev. B*, vol. 62, pp. 1102–1113, Jul 2000. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.62.1102

[38] H. Bethe, "Zur theorie der metalle," *Zeitschrift für Physik*, vol. 71, no. 3–4, p. 205–226, 1931.

[39] J. Sakurai and J. Napolitano, *Modern Quantum Mechanics.* Addison-Wesley, 2011. [Online]. Available: https://books.google.it/books?id=N4I-AQAACAAJ

[40] F. Becca and S. Sorella, *Quantum Monte Carlo approaches for correlated systems.* Cambridge University Press, 2017.

[41] A. M. Läuchli, C. Lacroix, P. Mendels, and F. Mila, *Numerical Simulations of Frustrated Systems.* Springer, 2011, p. 481–511.

[42] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *arXiv:1512.03385*, 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[43] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015. [Online]. Available: https://arxiv.org/abs/arXiv:1502.03167

[44] A. Messiah, *Quantum mechanics.* Dover Publications, 2014.

[45] J.-G. Liu, S.-H. Li, and L. Wang, *Lecture Note on Deep Learning and Quantum Many-Body Computation.* Chinese Academy of Sciences, 2018. [Online]. Available: https://wangleiphy.github.io/

[46] “Medicine laureates: Fields,” Nobelprize.org, Oct 2018. [Online]. Available: https://old.nobelprize.org/nobel_prizes/medicine/fields.html

[47] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, p. 433–460, Jan 1950.

[48] S. Vempala and J. Wilmes, “Polynomial convergence of gradient descent for training one-hidden-layer neural networks,” *arXiv:1805.02677*, 2018. [Online]. Available: https://arxiv.org/abs/1805.02677

[49] S. Sonoda and N. Murata, “Neural network with unbounded activation functions is universal approximator,” *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233–268, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1063520315001748

[50] R. Eldan and O. Shamir, “The power of depth for feedforward neural networks,” *arXiv:1512.03965*, 2015. [Online]. Available: https://arxiv.org/abs/1512.03965

[51] Y. S. John Duchi, Elad Hazan, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[52] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, Jun 1976. [Online]. Available: https://doi.org/10.1007/BF01931367

[53] D. Sussillo and L. F. Abbott, "Random walk initialization for training very deep feedforward networks," *arXiv:1412.6558*, 2014. [Online]. Available: https://arxiv.org/abs/1412.6558

[54] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[55] H. Fehske, R. Schneider, and A. Weisse, *Computational many-particle physics*. Springer, 2008.

[56] P. Mehta and D. J. Schwab, "An exact mapping between the variational renormalization group and deep learning," *arXiv:1410.3831*, 2014. [Online]. Available: https://arxiv.org/abs/1410.3831

[57] A. Fischer and C. Igel, "An introduction to restricted boltzmann machines," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, L. Alvarez, M. Mejail, L. Gomez, and J. Jacobo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 14–36.

[58] X. Gao and L.-M. Duan, "Efficient representation of quantum many-body states with deep neural networks," *Nature Communications*, vol. 8, no. 1, p. 662, 2017. [Online]. Available: https://doi.org/10.1038/s41467-017-00705-2

[59] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, pp. 625–660, Feb 2010.

[60] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 2006.

[61] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[62] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996. [Online]. Available: https://doi.org/10.1162/neco.1996.8.7.1341

[63] D. Liu, S.-J. Ran, P. Wittek, C. Peng, R. Blázquez García, G. Su, and M. Lewenstein, “Machine learning by two-dimensional hierarchical tensor networks: A quantum information theoretic perspective on deep architectures,” *arXiv:1710.04833*, 2017. [Online]. Available: https://arxiv.org/abs/1710.04833

[64] I. Glasser, N. Pancotti, and J. I. Cirac, “Supervised learning with generalized tensor networks,” *arXiv:1806.05964*, 2018. [Online]. Available: https://arxiv.org/abs/1806.05964