LUND UNIVERSITY

LUND UNIVERSITY

FACULTY OF SCIENCES

Division of Mathematical Physics
Bachelor thesis (15 hp)

Solving the Schrödinger equation with artificial neural networks

Author
Ibrohim HAMOUD

Supervisor
Andrea IDINI

June 29, 2021

Abstract

Recent implementations of artificial neural networks in solving the Schrödinger equation have realized significant advances regarding single-particle and many-body problems. Encouraged by the achievements in this subject, we train feed-forward neural network and RBF-network to find solutions to the one-dimensional Schrödinger equation. In this work, we consider a single-particle wavefunction in spatial coordinates. Firstly, The training of neural networks is implemented on various potentials in a supervised fashion. Afterward, this project introduces a variational representation of quantum states based on an artificial neural network with a variable number of hidden units. Thereby, we demonstrate that the Monte-Carlo method for a single-particle problem can be translated into a straightforward computational form aided by artificial neural networks. Throughout this document, the solutions of the Schrödinger equation obtained using neural networks are compared to the exact solutions. We begin this work with a full review of the relevant parts of quantum mechanics and artificial neural networks, followed by a quantitative discussion regarding results, consideration, and challenges.

Acronyms

SE Schrödinger Equation

ANN Artificial Neural Network

RBF Radial Basis Function

VMC Variational Monte-Carlo

SC Stochastic Reconfiguration

FFNN Feed-Forward Neural Network

ReLu Rectifier Linear Unit

FFT Fast Fourier Transform

Contents

Chapter 1

Introduction

The Schrödinger equation is one of the fundamental equations in modern physics. Conceptually, the Schrödinger equation is the quantum counterpart of Newton's second law in classical mechanics. The equation can be time-independent or time-dependent. The first case describes the stationary state of a quantum system, while the second case gives the evolution over time of a wavefunction. The Schrödinger equation relates to a many-particle quantum system, for example, it is used to predict the chemical and physical properties of a molecule.

Artificial neural networks present a new emerging powerful tool to solve the many-body problem. They form algorithmic function approximators capable of transforming a highly complex multivariate function into an approximate combination of simple univariate functions. They perform this approximation through optimization. What is unique about neural networks is that they can give a rough estimate to an unknown function simply by establishing a relation between input and output data without explicitly expressing the process that governs this relation between the data [1].

Although there is extensive research related to this field spanning several decades (cf. [2] with the first general application of neural networks as wavefunctions), most approaches were developed in recent years. This shows that interest in the topic has been increasing, due to recent results bridging the gap between neural networks and quantum mechanics. These novel approaches deliver highly accurate solutions to long-standing genuinely difficult single-particle and many-body problems [3][4]. In 2018 Teng showed how a radial basis function (RBF) network corresponds to a variational method in coordinate space and can find solutions to the one-body Schrödinger equation [5]. Regarding the quantum many-body problem, one of the first works relating artificial neural networks to Variational Monte-Carlo (VMC) was published by Carleo et al. in 2016 [6]. VMC is an optimization method used to find the ground state of a quantum system based on minimizing the energy expectation value (refer to [7]). This work was followed by extensive research regarding deploying neural networks through VMC [8][9]. Additional work focused on analyzing the potentials and limitations of the neural networks in the context of VMC [10][11]. These works focused on the Restricted Boltzmann Machine (RBM) as a method to represent the variational wavefunction with a neural network.

Feed-forward neural networks (FFNN), similarly to the RBM, were proven to be universal approximators [12]. However, it might occur that only certain neural network designs can represent a specific wavefunction because the neural network's size and shape put practical limits on the capacity of the network to achieve a good approximation. Therefore, whether a neural network with a simple architecture can define a function as the wavefunction is of particular interest. In general, a deep neural network can represent functions with polynomials of many parameters, while a shallow neural network needs to be expanded exponentially in terms of the number of hidden units to be able to approximate a certain function [13]. FFNNs are investigated intensively and adopted to solve the single-particles and many-body problems. In [14], the ground state is approximated through the VMC with an FFNN based trial wavefunction in continuous space. Similar work is done in the momentum space [15]. These works deal with the many-body problem and account for the exchange symmetry condition.

In summary, a variety of approaches were made regarding the application of neural networks in VMC that consider different quantum systems. However, an overall similarity holds between these approaches even though they differ conceptually. Instead of designing a problem-specific trial wavefunction for the VMC method, a novel class of very general trial wavefunctions based on neural networks are considered with relatively little physical insights required.

This thesis is organized as follows: In chapter 2, we review the theoretical background about solving the Schrödinger equation with the artificial neural network. Section 2.1 introduces the Schrödinger equation and demonstrates how to solve it as an eigenvalue problem using the finite difference method. In section 2.2, the VMC method is reviewed and associated with neural networks. A review of the basic principles of the neural network is given in section 2.3. Subsection 2.3.1 covers the technical details of constructing a neural network, mainly initialization. The results are presented with an amply discussion in chapter 3. Sections 3.1 and 3.2 provide the outcomes of training the neural networks on producing wavefunctions from input potentials and include a discussion about problem setup and the technical-related questions. The results for finding the ground state with VMC follow in section 3.3.

Chapter 2

Methods

2.1 The Schrödinger equation

The Schrödinger equation (SE) is a linear partial differential equation that describes a quantum system. Using the Schrödinger equation it is possible to obtain the eigenstates and energy levels of a quantum particle in a given potential. The equation's predicament is that it can be solved exactly only for the simplest of systems - such as the so-called toy models (e.g. particle in a box) and the Hydrogen atoms; and not for relatively complex systems. Therefore, the equation in general can only be solved numerically with an approximate solution. This is particularly true for many-particle systems, such as the Helium atom. An artificial neural network can be used as a numerical tool to reach such an approximate solution.

The general form for one dimensional time-independent Shrödinger equation for an arbitrary potential $V(x)$, reads

$$-\frac{\hbar^2}{2m}\frac{\partial^2\Psi(x)}{\partial x^2} + V(x)\Psi(x) = E\Psi(x). \tag{2.1}$$

The wavefunction $\psi(x)$ is a purely spatial function and it is a mathematical descriptor of the quantum state. Inserting the classical Hamiltonian function into equation (2.1) gives the following form,

$$\hat{H}\psi(x) = E\psi(x). \tag{2.2}$$

The wavefunction has a probability interpretation, where $|\Psi(x)|^2dx$ represents the probability of finding the particle in the interval dx, therefore the wave function should be normalized as follows,

$$\int_{-\infty}^{\infty} |\Psi(x)|^2 dx = 1. \tag{2.3}$$

For the normalization of the wavefunction we require for our case that the wavefunction should satisfy zero boundary condition, thus $\Psi(-\infty) = 0$ and $\Psi(\infty) = 0$.

Equation (2.2) has the form of an eigenvalue problem. The solutions to the equation are those functions that, when acted upon by the Hamiltonian operator, are left unchanged except for multiplication by a constant E which constitutes the energy of the state as an eigenvalue.

The equation can be numerically solved on a grid x_1, x_2, . . . , x_n for a finite length scale using matrix representation. The zero boundary condition is prescribed at the end points of the grid. Introducing the discretization of the second derivative

$$\left.\frac{d^2\Psi}{dx^2}\right|_{x_i} = \frac{\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}}{\Delta x^2}, \qquad i = 2, 3, \ldots, n-1,$$

the wave equation can be written as

$$\frac{-\hbar^2}{2m\Delta x^2}(\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}) + V_i\Psi_i = E\Psi_i, \qquad i = 2, 3, \ldots, n-1.$$

Introducing the column vector

$$\mathbf{\Psi} = (\Psi_2, \Psi_3, \ldots, \Psi_{n-1})^T$$

and the matrices

$$\hat{T} = \frac{-\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} = \frac{-\hbar^2}{2m\Delta x^2}\underbrace{\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \ldots & 0 & 0 & 0 \\ & \ddots & & & & \vdots & & \ddots & \\ 0 & 0 & 0 & 0 & 0 & \ldots & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 1 & -2 \end{pmatrix}}_{n-2\times n-2}$$

and

$$\hat{V} = \underbrace{\begin{pmatrix} V_2 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & V_3 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & V_4 & 0 & 0 & \ldots & 0 & 0 & 0 \\ & \ddots & & & & \vdots & & \ddots & \\ 0 & 0 & 0 & 0 & 0 & \ldots & 0 & V_{n-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & V_{n-1} \end{pmatrix}}_{n-2\times n-2},$$

in this case $\hat{V}$ represents a one-body local potential $V(x)$.

The discretized Schrödinger equation can be written in matrix form,

$$(\hat{T} + \hat{V})\mathbf{\Psi} = E\mathbf{\Psi}, \tag{2.4}$$

which is a symmetric eigenvalue problem, and it is formulated for a finite, orthonormal basis. The eigenvalues in equation (2.4) give the possible discrete energies of the particle, and the corresponding normalized eigenvectors $\{\psi_n(x)\}$ define the wavefunction on the entire grid with zero boundary condition imposed on the edges of the grid [16].

However, it is a frequent occurrence in practical calculations to adopt a non-orthonormal basis set instead. A paradigmatic example is the calculation of the electronic structure of molecules using atom-centered localized functions. In that case, equation (2.2) is reformulated to a generalized eigenvalue problem as follows,

$$\hat{H}\mathbf{\Psi} = ES\mathbf{\Psi}, \tag{2.5}$$

where S is the overlap matrix whose elements are given by $S_{mn} = \int_{-\infty}^{\infty} \psi_m(x)^* \psi_n(x) dx$ and it is an identity matrix for orthonormal basis. The Hamiltonian in (2.5) is constructed by evaluating the elements of the kinetic and potential energy matrices through

$$T_{mn} = \int_{-\infty}^{\infty} \psi_m^*(x) \hat{T} \psi_n(x) dx, \tag{2.6}$$

$$V_{mn} = \int_{-\infty}^{\infty} \psi_*^m(x) \hat{V} \psi_n(x) dx. \tag{2.7}$$

As a result, the Hamiltonian does not give a sparse matrix and needs to be diagonalized. In theory, it is always possible to derive an orthogonal basis set from a non-orthogonal one and reduce the problem to a symmetric eigenvalue problem [17]. In practice, Orthogonalization methods and converting the overlap matrix into an identity matrix are computationally expensive and not very stable numerically for large-scale eigenvalue problems.

In order to numerically solve a two- or three-dimensional Schrödinger equation, it is possible to use some generally applicable grid-based techniques for boundary value partial differential equations [18]. However, the accuracy of solutions is reduced for a low number of grid points, while matrices' size involved grows quadratically with the number of the grid points. On the other hand, the number of grid points grows exponentially with the dimensions. Thus, the large number of grid points required often makes such approaches too time-consuming and intractable. This is called the curse of dimensionality. Therefore, approximate variational methods are a frequent alternative in large-scale atomic, molecular, and solid-state physics calculations.

2.2 Variational Monte-Carlo (VMC)

The chief principle that validates VMC is the realization that many of the physical properties of a system can be determined mainly from the lowest energy eigenvalue and its corresponding ground state. The ground state contains information about most thermodynamic and equilibrium properties of the system at zero temperature, and additionally, many physical systems never depart far from their ground state. Thus, it often suffices to find the wavefunction that accommodates the lowest expected value of the energy to describe the system. On the other hand, VMC methods overcome the cost of explicitly solving the eigenvalue problem by translating it into an optimization problem. The outcome is an approximate solution with a sufficient representation of the ground state.

The variational principle states that given a system with a Hamiltonian $\hat{H}$, then if ψ is a well behaved, normalized wavefunction that satisfies the boundary conditions of the Hamiltonian, then

$$\langle\psi| \hat{H} |\psi\rangle \geq E_0, \tag{2.8}$$

where E_0 is the true value of the lowest energy eigenvalue of $\hat{H}$. The principle allows the calculation of an upper bound for the ground state energy by modifying a trial wavefunction ψ_T such that the expression of the functional $\langle\psi_T| \hat{H} |\psi_T\rangle$ is minimized. Hence, the trial wavefunction is varied until the expectation value converges into an optimal solution.

For wavefunctions that are not normalized, the variation integral for one dimension becomes

$$\mathbf{E}[\psi_T] = \frac{\langle\psi_T|\,\hat{H}\,|\psi_T\rangle}{\langle\psi_T|\psi_T\rangle} = \frac{\int \psi_T^*(x)\hat{H}\psi_T(x)dx}{\int \psi_T^*(x)\psi_T(x)dx} \geq E_0. \tag{2.9}$$

For a high number of dimensions, the high-dimensional integrals can be evaluated using Monte-Carlo-based techniques; hence, the method is called Variational Monte-Carlo (VMC).

To proceed with minimizing the expression of $\mathbf{E}$, the wave function is proposed to be dependent on a set of free parameters $\{\alpha\}$, dubbed the variational parameters, such that the wavefunction is varied with respect to these parameters for the objective of minimizing the energy expectation value. With the inclusion of the variational parameters, the energy expectation value is

$$\mathbf{E}[\psi_T] = \frac{\int \psi_T^*(x;\alpha)\hat{H}\psi_T(x;\alpha)dx}{\int \psi_T^*(x;\alpha)\psi_T(x;\alpha)dx} \geq E_0. \tag{2.10}$$

The variational ansatz in equation (2.10) can be rewritten as follows,

$$\mathbf{E}[\psi_T] = \frac{\int \frac{\psi_T(x;\alpha)}{\psi_T(x;\alpha)}\psi_T^*(x;\alpha)\hat{H}\psi_T(x;\alpha)dx}{\int \psi_T^*(x;\alpha)\psi_T(x;\alpha)dx} = \frac{\int |\psi_T(x;\alpha)|^2\frac{1}{\psi_T(x;\alpha)}\hat{H}\psi_T(x;\alpha)dx}{\int \psi_T^*(x;\alpha)\psi_T(x;\alpha)dx} \tag{2.11}$$

$$= \int P(x;\alpha)E_L(x;\alpha)dx \approx \frac{1}{N_{samples}}\textstyle\sum_i^N E_L(x_i;\alpha), \tag{2.12}$$

with N being the number of Monte-Carlo samples. $P(x) = \frac{|\psi_T(x;\alpha)|^2}{\int |\psi_T(x;\alpha)|^2}$ is defined as the probability distribution function and $E_L(x) = \frac{1}{\psi_T(x;\alpha)}\hat{H}\psi_T(x;\alpha)$ is the local energy. Given the probability distribution function, it is clear that the expectation energy can be evaluated statistically by a random walk in the configuration space $\{x_i\}$. This can be achieved with a variety of importance sampling methods such as a Markov chain Monte-Carlo (MCMC) that defines the probability of selecting and accepting the next configuration in order to generate a sequence of configurations $\{x_i\}_{i=1\ldots N_{sample}}$.

In order to minimize the expectation energy, the variational parameters in the variational wavefunction can be adjusted using the stochastic reconfiguration method (SC). The trial wavefunction is not orthogonal to the ground state, and by applying an operator $(\Lambda - \hat{H}) \leq 0$ with Λ being a sufficiently large number, the projected wavefunction $(\Lambda - \hat{H})\psi_\mathrm{T}$ is obtained. Thus, the SC method aims to adjust the variational parameters such that the new trial wavefunction is close to the projected one [19][20]. An ansatz of the new wavefunction is set equal to the Taylor expansion of the ψ_T as follows,

$$\psi_T^{new}(x;\alpha) = \delta\alpha_0\psi_T(x;\alpha) + \textstyle\sum_{j=0}^p \delta\alpha_j\frac{\partial\psi_T(x;\alpha)}{\partial\alpha_j}. \tag{2.13}$$

Applying the log derivative trick [21], equation (2.13) can be rewritten to be

$$\psi_T^{new}(x;\alpha) = \textstyle\sum_{j=1}^p \delta\alpha_j O_j(x), \tag{2.14}$$

where $O_j(x)$ is the logarithmic derivative, and it reads

$$O_j(x) = \frac{1}{\psi_T(x;\alpha)} \frac{\partial \psi_T(x;\alpha)}{\partial \alpha_j}. \tag{2.15}$$

For the variational parameters to be updated in SC, the covariance matrix S needs to be introduced. The elements of this matrix can be stochastically computed as follows,

$$s_{j,j'}(x) = \frac{1}{N_{samples}} \sum_i^{N_{samples}} [(O_j(x_i) - \overline{O_j})(O_{j'}(x_i) - \overline{O_{j'}})], \tag{2.16}$$

where $\overline{O_k}$ denotes the expectation value of the logarithmic derivative and it reads

$$\overline{O_k} = \frac{1}{N_{samples}} \sum_i^{N_{samples}} O_k(x_i). \tag{2.17}$$

Finally, the gradient descent is applied to update the variational parameters according to

$$\alpha_{j+1} = \alpha_j - \lambda \sum_i s_{i,j}^{-1} \frac{\partial E_L(x_i;\alpha)}{\partial \alpha_j} = \alpha_j - \tilde{\lambda} \frac{\partial \mathbf{E}[\psi_T]}{\partial \alpha_j}, \tag{2.18}$$

where λ and $\tilde{\lambda}$ are the update rate, j denotes the update time and S^{-1} is the inverse matrix of S. if S is not invertible, the inverse matrix is usually replaced with a pseudo-inverse matrix which is a generalization of the matrix inverse [22] . The matrix S represents the correlation between the different logarithmic derivatives $O_i(x)$, namely, the linear relation between them. The need to introduce the matrix S arises from that the variational wavefunction might end very differently when an update $\delta\alpha$ is applied on the parameter α_i than parameter $\alpha_{i'}$. This differentiation can lead to discriminating between identical wavefunctions that differ only in phase factor. In summary, the VMC involves the following steps,

1- Generating a sequence of spacial configurations $\{x_i\}_{i=1\ldots N}$ according to a probability distribution $P(x)$.
2- Computing the local energy $E_L(x_i)$ and the logarithmic derivatives $O_j(x_i)$ for all samples x_i and all the variational parameters α_j.
3- Computing the gradient descent $\frac{\partial \mathbf{E}[\psi_T]}{\partial \alpha_j}$ matrix elements $s_{j,i}$ for all the variational parameters α_j.
4- Updating the trial wavefunction $\psi_T(x;\alpha)$ and continue to converge into the exact wavefunction.

In neural network language, the variational parameters are represented by the connection strength between the neurons (weights). These weights are adjusted with classical machine learning optimization algorithms like Adam (Details regarding this algorithm are presented in appendix A). With neural network's optimization method, equation (2.18) takes the form,

$$\omega_{i+1} = \omega_i - \lambda \frac{\partial \mathbf{E}[\psi_T]}{\partial \omega_i}, \tag{2.19}$$

with ω being the set of weights in the neural network, and the energy derivative is propagated using a machine learning backpropagation algorithm (A full description of backpropagation algorithm is provided in section 2.3). The backpropagation can exploit the covariance matrix and adjust the learning rate accordingly. This applies to other minimization algorithms in machine learning as well. Therefore, weights update with backpropagation algorithm implicitly involves the above-mentioned VMC-steps ($1 \rightarrow 4$).

Critical to the outcome of a VMC calculation is a wise choice of the proposed trial wavefunction. To have definite and reliable results, the wavefunction should be a well-behaved function obeying the conditions of being normalizable, a continuous function of space with a continuous first-order spatial derivative, and it should abide by suitable symmetry requirements for the case of many-particle systems. These conditions stem from the requirement that the energy functional of the trial wavefunction is capable of approximating the actual underlying ground state when adjusting the variational parameters. However, an artificial neural network can be employed as a general-purpose trial wave function.

2.3 Artificial neural network (ANN)

Artificial neural networks based models are being used widely as an alternative method for solving ordinary and partial differential equations. Artificial neural networks utilize a processing strategy in which large numbers of computing units perform their calculations simultaneously. These units are called neurons. The enormous number of connections between these neurons can store various types of information. The solution to the differential equation in neural networks can be introduced as a superposition of piecewise linear functions that are represented by the neurons.

The advantages of using neural networks to solve differential equations are; (i) artificial neural networks are universal function approximators, (ii) the approximate neural networks-solutions of the differential equations are continuous over the domain of integration, (iii) a low number of model parameters are required [23]. The Schrödinger equation is a linear partial differential equation that had got attention as a problem where neural networks can be deployed.

There are many different types and topologies of neural networks. Feed-forward neural network (FFNN) is the earliest and most elementary type. In this network, the input data move forward from the input nodes throughout the hidden nodes to the output nodes. These nodes are arranged into different layers, with only inter-layer connections are considered. We examine only FFNN as a neural network for solving the Schrödinger equation in this work. Many other types such as recurrent neural networks are used extensively in the literature to solve the Schrödinger equation [24].

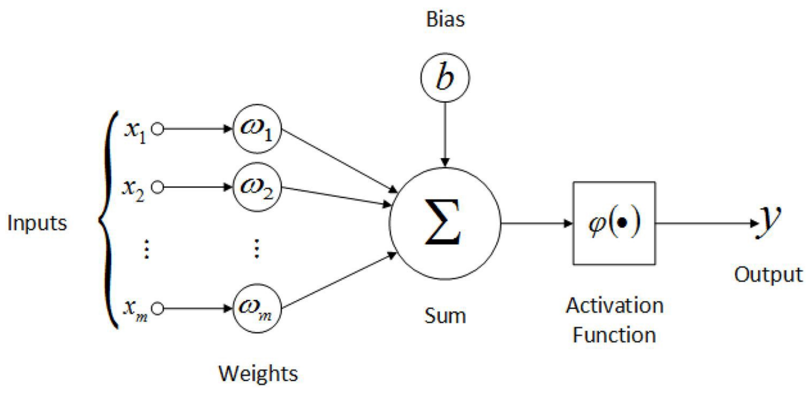

Figure 2.1: The basic element of the Artificial neural network, the neuron (from [25]).

The computation executed by the neuron is a weighted summation of its incoming signals and outputs a new signal that is a function of the input. The output of the neuron in figure 2.1 is calculated as,

$$y = \phi(\Sigma_{i=1}^{m}\omega_i x_i + b), \tag{2.20}$$

where $\{x_i\}$ is the input of the neuron, b is the bias, ω_i is the weight of the input x_i and the function $\phi(\bullet)$ is the activation function. There are several kinds of activation functions. The ones used in this work are Rectifier Liner Unit (ReLu), Softplus and Gaussian activation functions (cf. figure 2.2).

- $\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ \text{x} & \text{if } x > 0 \end{cases}.$
- $\text{Softplus}(x) = ln(1 + e^x)$.
- $\text{Gaussian}(x) = \frac{1}{\sigma 2\pi} e^{\frac{-x^2}{2\sigma^2}}$, for variance σ and mean $= 0$.

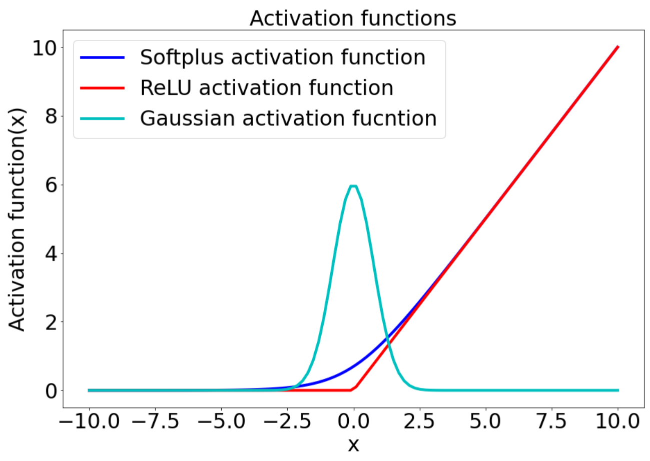

Figure 2.2: Plots of Softplus, ReLU and Gaussian activation functions.

An example of FFNN with two hidden layers is depicted in figure (2.3). In this type of network, the signals are entering from the left then passing on to the right. The number of hidden nodes for each layer can be arbitrarily assigned with a non-linear activation function alike to the ones mentioned above.

Given an input signal x_n and assuming one hidden layer, the output $y_i(x_n)$ is calculated through a forward pass $y_i(x_n) = \phi_y\Big(\Sigma_j \omega_{ij} h_{nj} + b_i\Big)$, where $h_{nj} = \phi_h\Big(\Sigma_k \tilde{\omega}_{jk} x_{nk} + b_j\Big)$. In one expression, the output is

$$y_i(x_n) = \phi_o\Big(\Sigma_j \omega_{ij} \phi_h\Big(\Sigma_k \tilde{\omega}_{jk} x_{nk} + b_j\Big) + b_i\Big). \tag{2.21}$$

This process can be iterated for an arbitrarily large number of layers. Thus, constructing a multi-layer neural network.

The learning algorithm relies on the backpropagation method based on computing the gradient of an error function and minimizing it. Assuming a training data set $\{x_n, d_n\}_{n=1..N}$ where d_{ni} denotes the target value for output i for signal n, the error function can be calculated from the mean square deviation

$$L(\omega) = \frac{1}{N}\Sigma_{n=1}^{N}\Sigma_i (d_{ni} - y_i(x_n))^2. \tag{2.22}$$

The error L is a function of the weights ω and is often minimized using the gradient descent method. For a multi-layer neural network, there are multiple sets of weights to compute weight update for using gradient descent. The weights update takes the form,

$$\Delta\omega_{ij}^{m} = -\eta \frac{\delta L}{\delta \omega_{ij}^{m}}, \tag{2.23}$$

where η is a hyper-parameter that defines the learning rate and m refers to the layer number.

Backpropagation is summarized in the following algorithm, assuming multiple layers and an input pattern to be A_i^0, where 0 stands for the input layer and i for the data input in the pattern. The signal flows forward in the network where the output of any nodes is given by,

$$A_i^m = \phi_i^m\Big(\Sigma_j \omega_{ij}^m A_j^{m-1} + b_i^m\Big).$$

When the signal reaches the output layer, the following function is defined and calculated,

$$\delta_i^M = \phi_i'^M\Big(\Sigma_j \omega_{ij}^M A_j^{M-1} + b_i^M\Big)\Big(d_i - A_i^M\Big),$$

where d_i is the target output from the node i and A_i^M is the actual output from the output layer M, and $\phi_i'^M$ is the derivative of the activation function in the output layer. This δ_i^m is calculated for each layer m propagating the error backward as follows,

$$\delta_i^m = \phi_i'^m\Big(\Sigma_j \omega_{ij}^m A_j^{m-1} + b_i^m\Big)\Sigma_j \omega_{ji}^m \delta_j^m.$$

Thus, the weights are updated with accordance to

$$\omega_{ij}^m(t+1) = \omega_{ij}^m(t) + \Delta\omega_{ij}^m, \text{ where } \Delta\omega_{ij}^m = \eta \delta_i^m A_j^{m-1}. \tag{2.24}$$

The presented algorithm uses on-stream updating, meaning that the weights are updated after each input pattern. In contrast, most networks use batch updating, meaning that the input pattern is accumulated to a specific size, and then the updating occurs at once for a batch of input patterns.

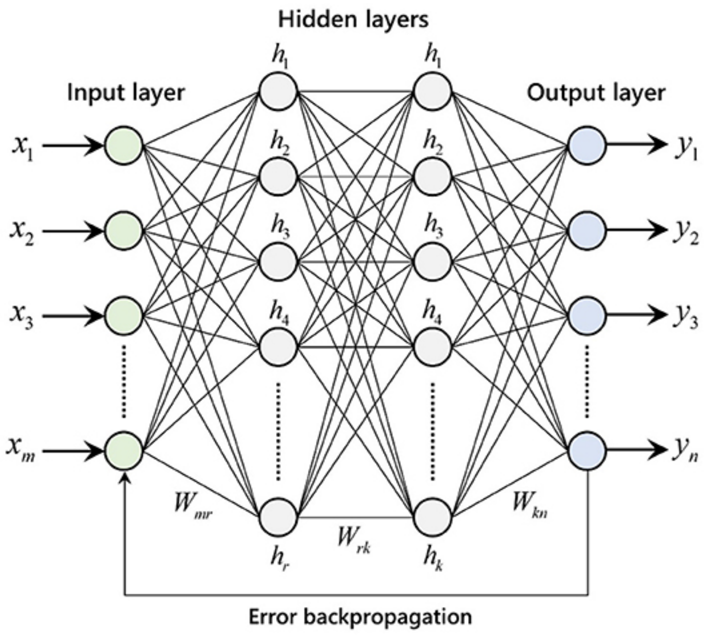

Figure 2.3: An illustration of an artificial neural network with the dots representing the network's nodes and the connections between the nodes depict the weights (from [26]).

2.3.1 Initialization of neural network

Initializing neural networks is an essential part of the training. Wrong initialization of a neural network can cause slow convergence to the target output or hamper weight updating. A good initialization is particularly significant for VMC. A well-initialized network can constitute a well-fitted trial ansatz and facilitate weights update during optimization. The importance of initialization is specific to the exploding and vanishing gradient problems. In a neural network of n layers, n derivatives will be multiplied together as the partial derivatives are found by chain rule traversing the neural network from the final layer to the initial layer. Therefore, if the derivatives are large, the gradients will risk increasing exponentially until they eventually explode. In contrast, if the derivatives are too small, the gradients will risk decreasing exponentially until they vanish. Additionally, the vanishing gradient might cause the initial layers to receive a minor gradient update. These first layers are updated slowly compared to other layers, which hampers the overall network's training. When the gradients vanish, it becomes challenging to know which direction the parameters should move to improve the loss function, and the training will slow down and even stops. On the other hand, exploding gradients make the learning unstable, and the loss changes drastically from update to update.

The exploding and vanishing gradient problem makes training deep neural networks a challenge despite all the known good practices used for initialization. Both cases can raise a problem dependent on the choice of the activation function. For example, an activation function that tends to be flat for large values gives the same output for an input of large variance. Finding the learning's theoretical optimum of the model relies heavily on the choice of the activation function as the derivative of the activation function is used during the optimization and affects the gradient. Low and large variance causes the gradient to move into a very narrow or broad range depending on the activation function, and both cases cause the neural network to learn very slowly. Therefore, The vanishing gradient is usually mitigated when using a ReLU-like activation function (Softplus activation function as an example) due to the shape of its derivative, which improves the gradient flow and does not cause small derivatives. In summary, Training neural networks can be challenging for several reasons: initialization, the activation function, and the architecture. These categories should be considered each to optimize the performance of the network.

Training neural networks can be particularly tricky when it is performed on mapping a highly complicated non-linear function, as is the case with solving the Schrödinger equation. Difficult problems might demand a very complex architecture that grants the network enough capacity to achieve learning. On the other hand, a very deep architecture has its drawbacks. The gradient diminishes fast, and the backpropagated errors become very small after a few layers making learning ineffective. Therefore, it is crucial to launch the network with proper weight initialization such that the learning is effective. On the other hand, dealing with many hidden layers puts the network constantly at the risk of over-fitting, where the neural network memorizes the patterns with the noise in the training data set. Thus, the neural network performs well on the training data set but fails to generalize to unseen data. In that case, the effect of over-fitting can be mitigated using proper regularization techniques such as L1, L2, and Dropout [27]. These techniques work on limiting the neural network capacity and constraining the model to reduce the risk of over-fitting .

Because selecting one initialization scheme over the other is based on the choice of an activation function, the He-initialization scheme is used and prioritized in this work over the Xavier-initialization scheme for its compatibility with the Softplus activation function. In [28], a rigorous mathematical proof is offered that for the ReLU-like activation function, the best weight initialization strategy is to initialize the weights randomly with the He-initialization scheme. However, no mathematical proof is available to match each specific activation function with a particular initialization scheme. Therefore, a good initialization scheme for complex and deep architecture is usually obtained using different algorithmic approaches supplemented with experimentation. An example of a simple algorithm for weight initialization can be demonstrated by Layer-sequential unit-variance (LSUV) initialization method [29]. Different initialization strategies were deployed in this work, and the He-initialization scheme showed to be the most compatible and gave the optimal model convergence.

Chapter 3

Results and discussion

3.1 Results for potential-density mapping using FFNN

This section presents the results of training different neural network architectures and assessing their performance in solving the Schrödinger equation. The plots of the neural network's performance are presented, as well the analysis of the parameters that are required to evaluate the performance of the said networks.

For the first part of the work, the neural networks are trained using supervised learning. The task of learning is to fit the neural network's weights such that the desired outputs are generated from the input data set. The training data set consists of potential-probability density function pairs with the density functions found by solving equation (2.4) numerically as an eigenvalue problem. Here, The Schrödinger equation is solved for a large set of harmonic oscillator potentials, simple inverted Gaussian potentials, and random potentials generated from a linear combination of inverse-Gaussian potentials. Only the probability density of the ground state wavefunction is considered and set as a target output for the neural networks. Thus, the neural network will be performing the task of potential-to-density mapping. In machine learning language, this process is treated as a class of sequence-to-sequence mapping as both potential and density have profiles of sequential data. In figures 3.1, the lowest six bound wave functions and the probability densities for an example-potential are presented.

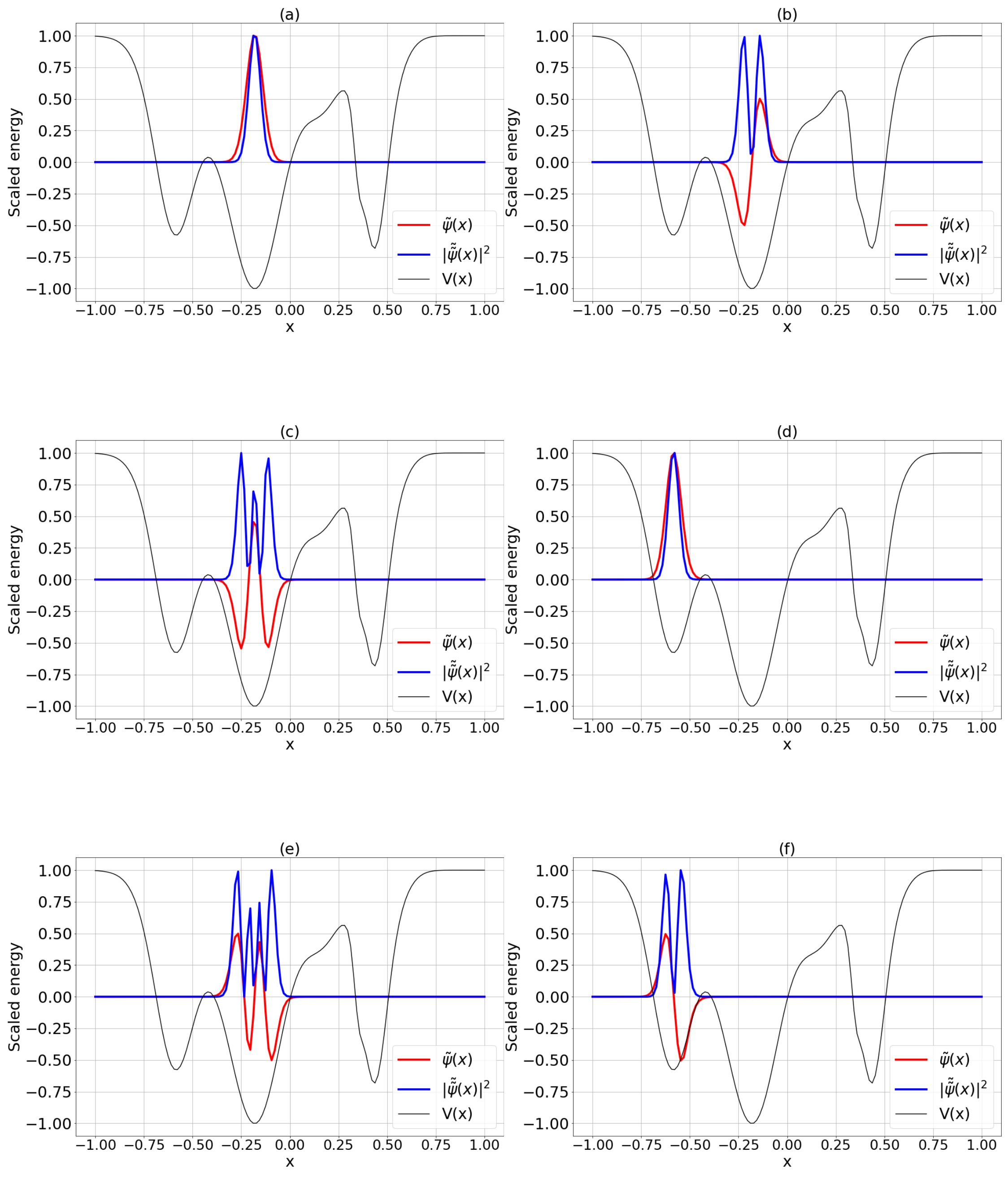

Figure 3.1: Plots of the six lowest bound wavefunctions and probability functions, for an example-potential generated from a linear combination of inverse-Gaussian potentials. For a better representation and to fit the plots in the same figure, the potential, the wavefunction, and the probability function of each bound state are scaled to have 1 as maximum value as follows, $V_i = 1 + \frac{2V_i}{V_{max} - V_{min}}$, $\tilde{\psi}_i = \frac{\psi_i}{\psi_{max} - \psi_{min}}$ and $|\tilde{\tilde{\psi}}_i|^2 = \frac{|\psi_i|^2}{|\psi_{max}|^2 - |\psi_{min}|^2}$, where the subscript i denotes the value of the function on the grid point, $\tilde{\psi}_i$ is the rescaled wavefunction and $|\tilde{\tilde{\psi}}_i|^2$ is the rescaled probability. It is important to note that as a result of the rescaling $|\tilde{\psi}_i|^2 \neq |\tilde{\tilde{\psi}}_i|^2$. The x-axis represents the spatial coordinates in nm unit, and the y-axis is a dimensionless scale.

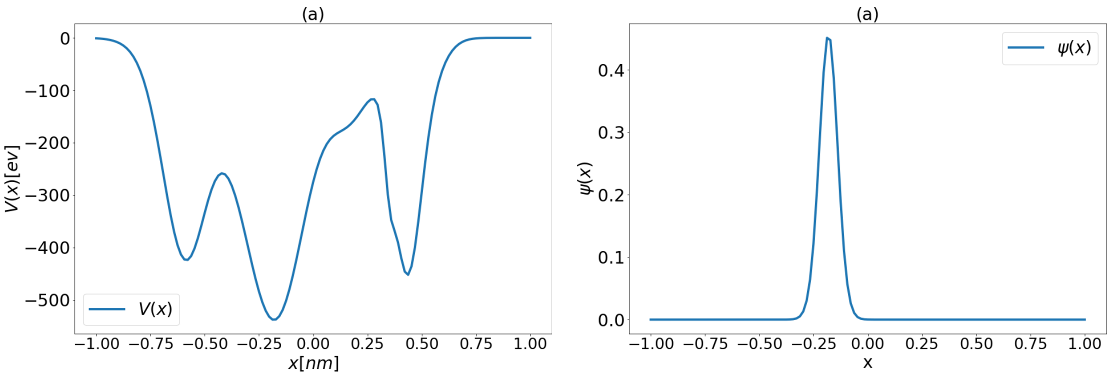

Figure 3.2: Plots of an example-potential generated from a linear combination of inverse-Gaussian potentials and its corresponding ground state wavefunction that are used in the data set to train the neural networks.

In order to approximate function $|\Psi(x_1, x_2, ..., x_n)|^2$ from $V(x_1), V(x_2), ..., V(x_n)$ as n independent input variable, a three-layer neural network is selected. To match the number of the input variables, the network is designed with n input neurons, h^1 hidden neurons by Softplus activation function and n output neurons by Softplus activation function. The input layer is made of source nodes and works as an information recipient, while the hidden layers perform a nonlinear transformation from input space to high dimensional hidden space. Thus, equation (2.21) can be written as follows, $|\Psi(x_r)|^2 \approx f^2(\Sigma_{j=1}^{h^1}\omega_{jk}^2 f^1(\Sigma_{i=1}^{n}\omega_{ij}^1 V(x_i) + b_j^1) + b_k^2)$, where b_t^s refers to the bias in the neuron i in the layer k.

To keep things simple, the neural network is trained for input data set made of a combination of simple inverted Gaussian potentials and harmonic oscillators of different depth, width, and position on the grid line. For this task, the network is trained on data set of 200000 examples of potential-density pairs, and the performance is checked on a different data set of 40000 examples. The network performed well on both training and validation data sets with $> 99\%$ accuracy. Figure 3.3(a). High accuracy was reached even for a shallow network of two hidden layers with 20 nodes for each layer.

Accuracy is one of the metrics to measure the performance of the model, and it is defined as

$$\frac{\text{Number of output nodes that satisfy the condition } \left||\psi_i^{target}|^2 - |\psi_i^{output}|^2\right| < 10^{-8}}{\text{Total number of output nodes}}, \quad (3.1)$$

where $|\psi_i|^2$ is the value of the density function on the output node i. For a batch size N, the number of output nodes that satisfy the condition mentioned above is summed over the samples and divided by N. It is important to emphasize that accuracy is used primarily to measure the efficacy and the performance of the neural, and it is not an indication of how accurate the network's prediction is. The accuracy is given in the form of a percentage on a scale from 0 to 1. For a single pattern example, an accuracy of 0.9 means that the model achieved correct predictions on 90% of the output nodes with disregard for the errors made on the remaining nodes; every error has the same weight. In comparison, the loss metric measures the distance between the target output and the actual output, thus how well the model's predictions averaged on all the output nodes. The distinction can be illustrated by the following case where the model gives high accuracy and a large loss, which means a very bad prediction on few output nodes. The loss metric is defined in this project as the logarithm of the loss function in equation (2.22), where a perfect prediction is equivalent to a loss function of zero value.

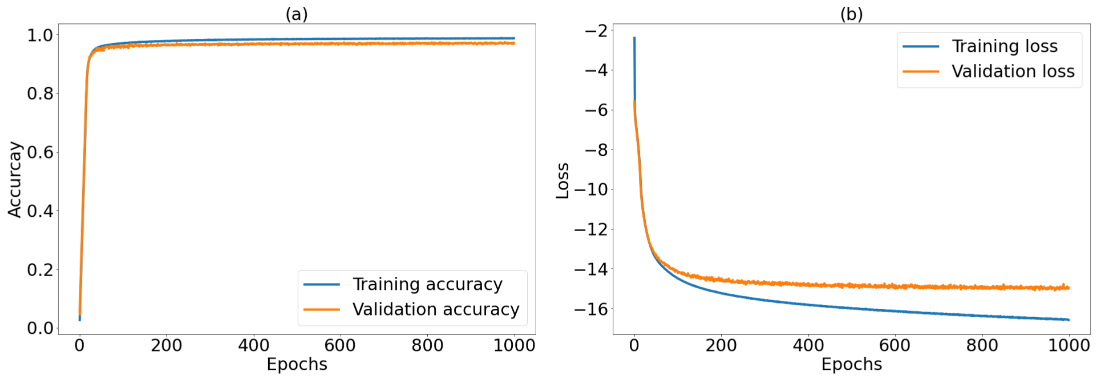

Figure 3.3: Accuracy and loss are plots for both training and validation data sets over a thousand epochs. The epoch number is the number of times the entire data set is passed through the neural network for training and validation. These results are given for a data set made of a combination of simple inverted-Gaussian potentials and harmonic oscillator potentials. Accuracy is defined in equation (3.1), and the loss is defined by the logarithm of the loss function in equation (2.22). The loss relates to how much the predicted value is close to the actual value.

On the other side, training the neural network proved to be more difficult for more complex potential-to-density mapping, such as a random potential built from a linear combination of different inverse-Gaussian potentials. The neural network performance showed a constant under-fitting on this problem, which requires increasing the network's capacity. Therefore, the neural network is expanded to 3 hidden layers with as many as 500 neurons. While adding a large number of layers is unproductive as the extra layers perform identity transformation, which renders these layers redundant. The problem of vanishing gradient appears with the inclusion of more layers. Therefore, the weights were initialized in the range $-1/N$ to $1/N$. This is to keep the variance of the weight values the same between every two layers. Herewith, the performance could reach an accuracy of 80%. However, after implementing the He-uniform initialization scheme, the network performance improved to 96% on the training set and 97.5% on the validation set. Over-fitting occurred when training for a very large epoch number (The number of times the learning algorithm works through the data set) and could be mitigated with the L2-regularization technique [27]. Overall, an epoch number larger than 1000 did not enhance the network's performance. A smaller batch size of around 300 to 500 showed to be crucial for a good performance because the model makes more frequent updates to the parameters. The neural network performance scored better on the validation data set than the training data set, which is attributed to using the Dropout technique during training. Dropout is applied to assist in reducing over-fitting . As a result, the neural network operates at a full capacity during the validation and uses all the features from the input data as opposed to the training.

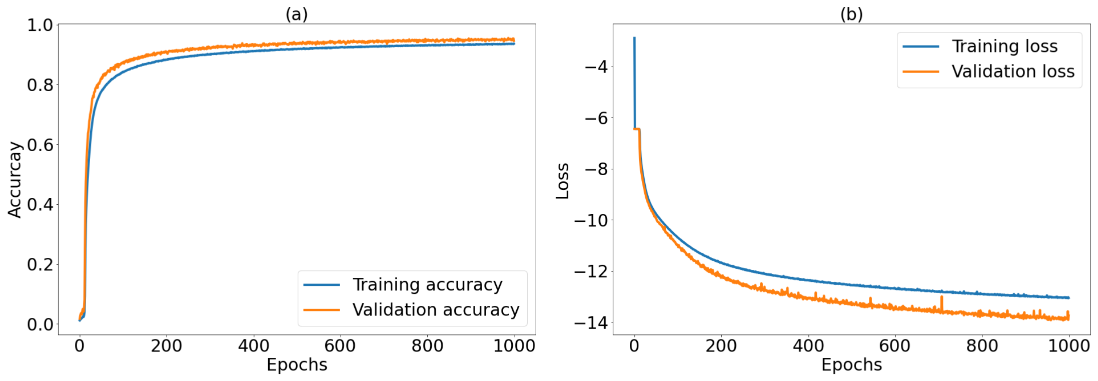

Figure 3.4: Accuracy and loss plots for both training and validation data sets with He-initialization scheme, solved for a linear combination of inverse-gaussian potentials.

It is worth noting that the first few trials for training the previous neural network did not score an accuracy above 90%. An accuracy close to one presented in the figure 3.4(a) was possible only after deploying several standard pre-processing techniques that made the input potential more comprehensive to the neural network. Mainly, standardisation and Fast Fourier Transform (FFT) of the input potential were used here. Standardization refers to re-scaling the input data since a large spread of input values can lead to large error gradient values, thus a dramatic weight change. Additionally, large input values can result in a model that learns large weight values, which renders the learning unstable. The standardization technique that is followed in this work is given by,

$$V_i = \frac{V_i - V_{mean}}{\sigma(V)}, \tag{3.2}$$

where $\sigma(V)$ denotes the standard deviation of the potential's values on the grid in respect to the mean value. On the other hand, pre-processing via FFT of the input potential aids the neural network with feature extraction and the detection of abnormal input patterns. For maximum utility, The neural network is fed with both standardized potential and its FFT as an input, which turns the input layer double in size. This method proved to be more efficient for training the neural network than feeding the neural network with either standardized potential or the FFT of the potential alone and yielded better accuracy on both the training and validation data sets.

The primary purpose of data pre-processing is to modify the input variables for a better match with the predicted output. Pre-processing can be thought of as a data transformation such that the most prominent features of the input data are highlighted and thus facilitating the learning process. There are other pre-processing strategies than standardization and FFT that are available but were not investigated as the one used here proved to be sufficient [30][31].

3.2 Results for potential-density mapping using RBF-netwrok

It is vital to have a good representation of the Gaussian tails of the wave function, which are usually located in the classically forbidden region. These tails are of particular interest because it is there where many of the physical processes occur. Good accuracy in representing the tails is essential for studying barrier penetration, nucleon's interaction with the nuclear shell, and radioactive decay.

The tails of the wavefunction have an exponential decay with near-zero value on the edge of the tails. Thus, a high overall accuracy scored by the network does not validate a good accuracy for the tails. To grant the network a better grasp of the wavefunction's tails, the loss function in equation (2.22) is modified accordingly,

$$L(\omega) = \frac{1}{N}\Sigma_{n=1}^{N}\Sigma_i(log(|\Psi_{ni}^{target}|^2) - log(|\Psi_{ni}^{output}|^2))^2. \tag{3.3}$$

To have better accuracy on the tails, the previous architectures are trained using radial basis activation function (RBF) for the hidden layer (the one used here is the Gaussian activation function defined in section 2.3). The net input to the RBF activation function is the vector distance between the weight and input vectors, multiplied by the bias. The RBF function has an output ≤ 1, and it increases as the distance between the weight and the input decreases. Therefore, the RBF function acts as a detector that produces one whenever the input vector and weight vector are identical. At the same time, the bias allows the sensitivity of neurons to be adjusted [5]. This architecture makes use of the radial basis method properties for solving partial differential equations [32]. These properties offer a more smooth approximation and accurate representation of the function than the polynomial spline fitting, which is accomplished by the previous architecture.

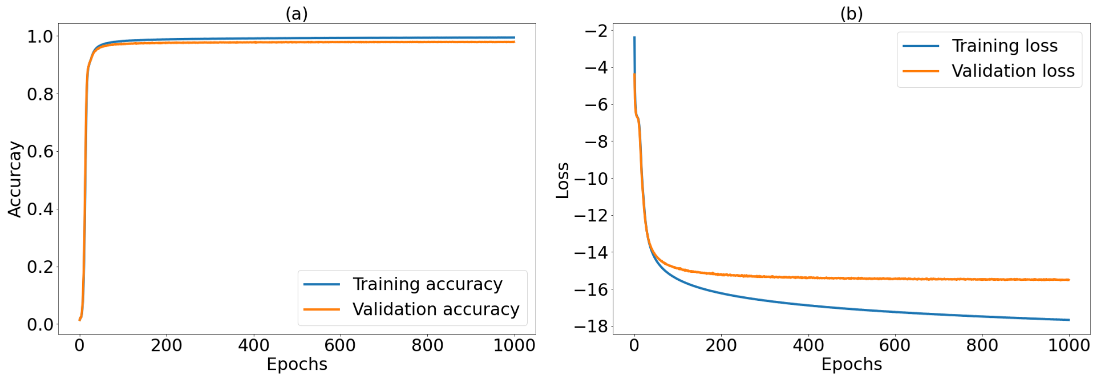

Figure 3.5: Accuracy and loss plots for both training and validation data sets using RBF-network. The results are given for harmonic oscillator potentials. Accuracy is defined from equation (3.1), and The loss is defined by the logarithm of the loss function in (3.3)

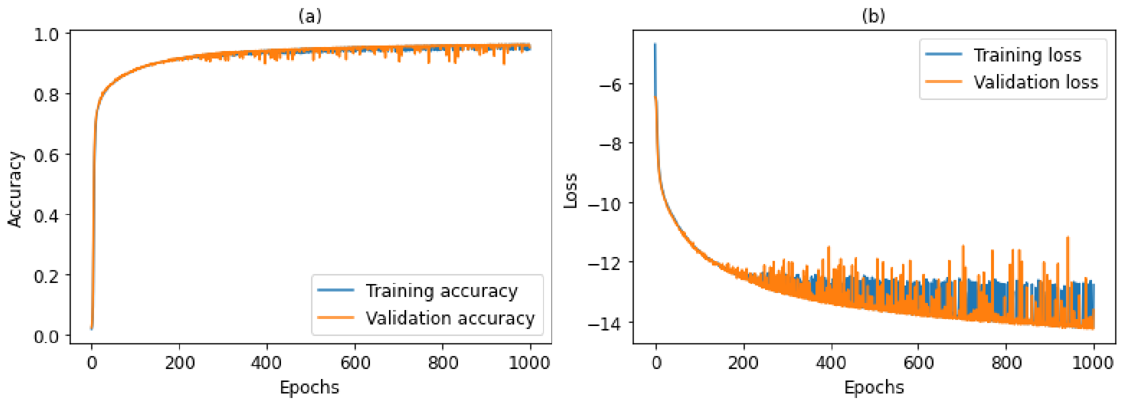

Figure 3.6: Accuracy and loss plots for both training and validation data sets using RBF-network. The results are given for a linear combination of inverse-Gaussian potentials. The fluctuation in the accuracy and loss plots is due to a large learning rate. Also, A possible cause can be that the regularization is insufficient for the size of the network.

3.3 Results from the variational method

Inspired by the universal approximation theorem, artificial neural networks are adopted to find the ground state energy through the variational method. RBF-network can be utilized as a general-purpose trial wavefunction of a continuous one-body and many-body system. Generally, the variational approach involves the postulation of a trial wavefunction, which demands physical insights into the quantum system in the question. The drawback of choosing a trial wavefunction is that it can be a difficult task and very problem-specific. This problem appears, for example, in determining a trial wavefunction to investigate energy metal-insulator transition in solid hydrogen using VMC [33]. Therefore, using the neural networks as a general-purpose trial wavefunction reduces the demand on the physical intuition and the inflexibility of the conventional VMC. The physical comprehension for designing the trial wavefunction is replaced with the question about the proper design of the neural network.

The neural network of choice is the RBF-netwrok with one hidden layer and six hidden units. The Gaussian activation function is selected for the hidden units and the Softplus activation function for the output layer. Given smooth activation functions, the network provides a smooth output function, which is expected from a physical wavefunction. The network is fed with input potential array $V_n = V(x_n)$, which represents particle location, and the weights of the network constitute the variational parameters. From equation (2.21) the trial wavefunction is given by,

$$\Psi_k(x_r; \{\omega\}) \approx f_o(\Sigma_j \omega_{jk}^{(2)} f_h(\Sigma_i \omega_{ij}^{(1)} V(x_i) + b_j^{(1)}) + b_k^{(2)}). \tag{3.4}$$

The expression of the energy functional in equation (2.10) is the cost function that is to be minimized. For one body problem, the Hamiltonian is given to be $\hat{H} = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]$. The variational parameters are adjusted incrementally over many iterations. The energy expectation value of each optimization step is compared to the exact energy to analyze the convergence and efficiency of the neural network. The neural network contains up to a thousand parameters that need to be adjusted to approximate the ground state wavefunction. The VMC optimization used is Adam-based, that is, a gradient-based optimization algorithm. This implies that some statistical errors are present in computing the gradient as well as discretization error in computing the cost function.

The weights are initialized with the He-initialization scheme, while the biases are initialized with random uniform distribution from -1 to 1. The choice of bias initialization is arbitrary and follows the standard practices for bias initialization. The learning rate is chosen to be in the soft point 0.001 for an optimum minimization of the cost function. A small learning rate prolongs the optimization of the weight parameters unwarrantedly. Conversely, a large learning rate leads to a bad convergence and an oscillating final value of the energy functional.

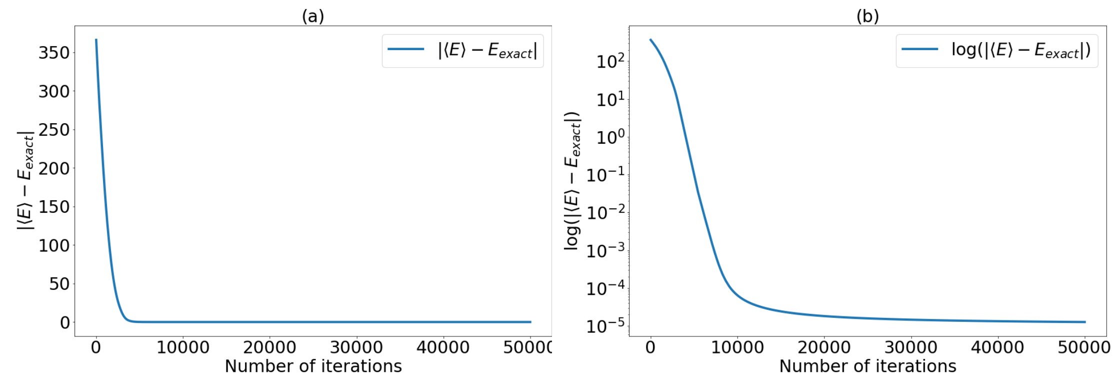

Figure 3.7: Plots (a) and (b) depict an example of the energy expectation value's convergence to the exact energy. The example is given for a linear combination of inverse-gaussian potentials. After 50 thousands of iterations, the expectation value shows a good agreement with the exact value with an accuracy of up to 6 decimal digits.

The convergence of the expectation value toward the exact value was observed to halt after a certain number of iterations. Part of the explanation for this halt is the truncation error of the numerical integration used to compute the energy functional. Computing the integral with a high precision showed to be critical for error minimization, which is defined here as the difference between the expectation value and the exact value. The task of numerical integration was achieved with composite closed Newton-Cotes quadrature rules [34][35]. This method is based on evaluating the integrand of equally spaced grid points, and therefore it is adequate for the task in question. For a grid of $n+1$ points, The formula is given by,

$$\int_a^b f(x)dx = \Sigma_{i=0}^{n}\omega_i f(x_i) = \Sigma_{i=0}^{n}\Big(\int_a^b l_i(x)\Big)f(x_i), \tag{3.5}$$

where ω_i are called weights and can be computed as the integral of Lagrange basis polynomials $l_i(x)$ of an arbitrary degree n.

In this work, the Newton-Cotes rule was applied for different degrees from degree 2 and as high as 20 degrees. While the precision of the expectation values increases for higher degrees, a plummet in the accuracy occurs for Newton-Cotes of degree 8 and higher, which can be attributed to Runge's phenomenon [36]. This problem was mitigated using the composite version of the lower degree Newton-Cotes rule, which gave the best results with the expectation value approaching the exact value up to 6 decimal digits of precision.

Another numerical integration that was investigated is Romberg's algorithm that gave better results for high extrapolation number [37]. However, computing the integrals using this algorithm over thousands of iterations proved to be more time-consuming and computationally costly. Therefore, the method is omitted in this work. For better precision, the prime choice of numerical integration method would be the Gauss–Legendre quadrature [38][39]. Nevertheless, the employment of this method appears to be unfeasible in our work as the method is based on calculating the integral of Legendre polynomials on a non-equidistant grid. Although, the deployment of the Gauss-Legendre rule can be workable if the roots of the Legendre polynomial integral are assigned to be the spatial coordinates or the grid points of the potential and the wavefunction. In this way, the potential and the wavefunction represented by the network's output are redefined to be functions of the Legendre polynomial integral's roots. Such a workaround warrants further investigation if better precision is required.

It should be noted that in the first training trials, the neural network was fed with the potential as an input. However, it was realized that feeding input to the neural network was not necessary in the first place, and the neural network functions well as a representation of the trial wavefunction with no input layer at all. Lifting the constraint of an input potential showed no adverse impact on optimizing the parameters and minimizing the energy functional.

With the success of obtaining an approximate ground wavefunction on a single potential, an attempt was made to train the neural network on different potentials and minimize different energy functionals. This exercise intends to model the network such that it can be a reliable representation of multiple ground state wavefunctions simultaneously. Such a multi-representative network can replicate the learning characteristics of the networks in section 3.2 but in an unsupervised manner. In essence, if the optimization of the weights proceeds on a large number of energy functionals simultaneously, the network's weights would converge to universal values capable of achieving potential-wavefunction mapping with no training data set. However, constructing a multi-representative network appears to be inaccessible. The neural network shows to approximate only one ground state at one time due to the weights readjusting themselves to meet the request of minimizing specific energy functional, and the information from any previous operation is lost. An alternative would be to steer different operational graphs simultaneously, meaning that the Adam optimization algorithm is run on different energy functionals and minimizing all of them at once. This procedure proved to be insufficient when using only a few energy functionals. On the other hand, training the neural network to minimize a large number of energy functionals involves defining a large number of operations. This process is strenuous on a technical level and computationally costly, and therefore it was not considered.

Additionally, it is postulated that achieving a multi-representative network requires expanding the network in terms of the number of hidden units. As was shown in sections 3.13.2, achieving potential-wavefunction mapping demands a large number of hidden units. The ramification of such an expansion is a large number of variational parameters.

Thus, the neural network would be sanctioned by insufficient optimization as it is challenging to keep track of all the parameters when performing VMC. This issue appears to be common in the literature regarding VMC-networks [40]. Therefore, an essential feature of these networks is their expressibility, meaning a capacity to represent a wavefunction with high accuracy using a moderate number of weight parameters. So far, notable work has been done to secure this property, for instance, developing VMC-specific optimization algorithms to tackle the problem of a large number of parameters and increase the network's expressibility. An example of such algorithms is the importance sampling gradient optimization algorithm (ISGO) [41].

Chapter 4

Conclusion

Many documents and research papers are emerging each year with novel and new implementations of artificial neural networks in quantum mechanics. The advances in this subject constitute the prime motivation for our work. Currently, artificial neural networks show performance comparable to previous state-of-the-art variational ansatzes for single-particle and many-body problems, which makes this subject an excellent topic for investigation.

This thesis provides solutions to the Schrödinger equation based on supervised learning and VMC. We demonstrate how neural networks can successfully find the relationship between the potential and the wave function. Our contribution manifests in optimizing a simple feed-forward neural network to obtain the best results. We implement pre-processing techniques and feed the neural network with the pre-processed potential and the actual potential jointly. Furthermore, we emphasize an educated choice of weight initialization scheme that is problem-relevant and fits the network architecture. This is followed by an amply discussion on how the network's parameters impact our results.

This work proceeds into incorporating a simple feed-forward neural in VMC for a single-particle problem. This approach relies on representing the variation wavefunction with the neural network using the standard machine learning optimization algorithm to find the ground state wavefunction. The method provides a positive outcome, and the convergence to the exact solution is satisfactory. We demonstrate the validity of our results by extending the discussion upon improving the accuracy. Our results show that neural networks are worth investigating to find solutions and implement many-body methods.

4.1 Outlook

On top of the many areas included in this work, there is much room for further research. We want to emphasize that investigating VMC for many-body problems using neural networks is the most compelling area to continue this work. Our approach might suffer from limitations regarding systems of many fermions due to the difficulty of imposing anti-symmetrization in continuous space. Thereby, we hypothesize that our approach is still valid if the row coordinates are replaced with Slater determinants to enforce antisymmetry. In addition, We limited our work to FFNN. However, investigating other neural network types might yield similar or better results, and it is worth exploring.

Bibliography

[1] S. Liang and R. Srikant, “Why deep neural networks for function approximation?,” *arXiv preprint arXiv:1610.04161*, 2016.

[2] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural network methods in quantum mechanics,” *Computer Physics Communications*, vol. 104, no. 1-3, pp. 1–14, 1997.

[3] J. Hermann, Z. Schätzle, and F. Noé, “Deep-neural-network solution of the electronic schrödinger equation,” *Nature Chemistry*, vol. 12, no. 10, pp. 891–897, 2020.

[4] D. Pfau, J. S. Spencer, A. G. Matthews, and W. M. C. Foulkes, “Ab initio solution of the many-electron schrödinger equation with deep neural networks,” *Physical Review Research*, vol. 2, no. 3, p. 033429, 2020.

[5] P. Teng, “Machine-learning quantum mechanics: Solving quantum mechanics problems using radial basis function networks,” *Physical Review E*, vol. 98, no. 3, p. 033305, 2018.

[6] G. Carleo and M. Troyer, “Solving the quantum many-body problem with artificial neural networks,” *Science*, vol. 355, no. 6325, pp. 602–606, 2017.

[7] W. Foulkes, L. Mitas, R. Needs, and G. Rajagopal, “Quantum monte carlo simulations of solids,” *Reviews of Modern Physics*, vol. 73, no. 1, p. 33, 2001.

[8] J. Chen, S. Cheng, H. Xie, L. Wang, and T. Xiang, “Equivalence of restricted boltzmann machines and tensor network states,” *Physical Review B*, vol. 97, no. 8, p. 085104, 2018.

[9] Y. Huang and J. E. Moore, “Neural network representation of tensor network and chiral states,” *arXiv preprint arXiv:1701.06246*, 2017.

[10] X. Gao and L.-M. Duan, “Efficient representation of quantum many-body states with deep neural networks,” *Nature communications*, vol. 8, no. 1, pp. 1–6, 2017.

[11] Z. Cai and J. Liu, “Approximating quantum many-body wave-functions using artificial neural networks,” *Physical Review B*, vol. 97, p. 035116, 2018.

[12] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[13] G. F. Montúfar, “Universal approximation depth and errors of narrow belief networks with discrete units,” *Neural computation*, vol. 26, no. 7, pp. 1386–1407, 2014.

[14] J. Kessler, F. Calcavecchia, and T. D. Kühne, “Artificial neural networks as trial wave functions for quantum monte carlo,” *Advanced Theory and Simulations*, vol. 4, no. 4, p. 2000269, 2021.

[15] J. Keeble and A. Rios, “Machine learning the deuteron,” *Physics Letters B*, vol. 809, p. 135743, 2020.

[16] R. Becerril, F. Guzmán, A. Rendón-Romero, and S. Valdez-Alvarado, “Solving the time-dependent schrödinger equation using finite difference methods,” *Revista mexicana de física E*, vol. 54, no. 2, pp. 120–132, 2008.

[17] B. N. Parlett, *The symmetric eigenvalue problem.* SIAM, 1998.

[18] J. Thijssen, *Computational physics.* Cambridge university press, 2007.

[19] S. Sorella, “Generalized lanczos algorithm for variational quantum monte carlo,” *Physical Review B*, vol. 64, no. 2, p. 024512, 2001.

[20] S. Sorella, “Green function monte carlo with stochastic reconfiguration,” *Physical review letters*, vol. 80, no. 20, p. 4558, 1998.

[21] C. J. Walder, P. Roussel, R. Nock, C. S. Ong, and M. Sugiyama, “New tricks for estimating gradients of expectations,” *arXiv preprint arXiv:1901.11311*, 2019.

[22] J. C. A. Barata and M. S. Hussein, “The moore–penrose pseudoinverse: A tutorial review of the theory,” *Brazilian Journal of Physics*, vol. 42, no. 1-2, pp. 146–165, 2012.

[23] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[24] C. Wang, H. Zhai, and Y.-Z. You, “Emergent schrödinger equation in an introspective machine learning architecture,” *Science Bulletin*, vol. 64, no. 17, pp. 1228–1233, 2019.

[25] J. B. Ahire, “The artificial neural networks handbook: Part 4,” *Medium. https://medium. com/@ jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e*, 2018.

[26] B. M. P. Pedro L. Fernández-Cabán, Forrest Masters, “Predicting roof pressures on a low-rise structure from freestream turbulence using artificial neural networks,” *Frontiers in Built Environment*, vol. 4, no. 68, 2018.

[27] M. A. Nabian and H. Meidani, “Physics-driven regularization of deep neural networks for enhanced engineering design and analysis,” *Journal of Computing and Information Science in Engineering*, vol. 20, no. 1, 2020.

[28] S. K. Kumar, “On weight initialization in deep neural networks,” *arXiv preprint arXiv:1704.08863*, 2017.

[29] D. Mishkin and J. Matas, “All you need is a good init,” *arXiv preprint arXiv:1511.06422*, 2015.

[30] N. M. Nawi, W. H. Atomi, and M. Z. Rehman, “The effect of data pre-processing on optimized training of artificial neural networks,” *Procedia Technology*, vol. 11, pp. 32–39, 2013.

[31] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, “Data preprocessing for supervised leaning,” *International Journal of Computer Science*, vol. 1, no. 2, pp. 111–117, 2006.

[32] Y. Shirvany, M. Hayati, and R. Moradian, “Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations,” *Applied Soft Computing*, vol. 9, no. 1, pp. 20–29, 2009.

[33] F. Calcavecchia and T. D. Kühne, “Metal-insulator transition of solid hydrogen by the antisymmetric shadow wave function,” *Zeitschrift für Naturforschung A*, vol. 73, no. 9, pp. 845–858, 2018.

[34] P. A. A. M. Junior and C. A. Magalhaes, “Higher-order newton-cotes formulas,” *Journal of Mathematics and Statistics*, vol. 6, no. 2, pp. 193–204, 2010.

[35] M. M. Gracca, “A simple derivation of newton-cotes formulas with realistic errors,” *arXiv preprint arXiv:1202.0237*, 2012.

[36] K. Adecalon, G. Nobeen, L. Nithoo, J. Girish, R. Jheengut, N. Ramburuth, S. Navinam, and P. Krisen, “Presented by,”

[37] T. Ergenç, I. Altas, *et al.*, “Romberg integration: A symbolic approach with mathematica,” in *International Conference on Computational Science*, pp. 691–700, Springer, 2003.

[38] R. McClarren, *Computational nuclear engineering and radiological science using python.* Academic Press, 2017.

[39] J. A. Trangenstein, *Scientific Computing, Vol. III – Approximation and Integration.* Springer International Publishing AG, 2018.

[40] E. Neuscamman, C. Umrigar, and G. K.-L. Chan, “Optimizing large parameter sets in variational quantum monte carlo,” *Physical Review B*, vol. 85, no. 4, p. 045103, 2012.

[41] L. Yang, Z. Leng, G. Yu, A. Patel, W.-J. Hu, and H. Pu, “Deep learning-enhanced variational monte carlo method for quantum many-body physics,” *Physical Review Research*, vol. 2, no. 1, p. 012039, 2020.

[42] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[43] Y. Wang, P. Zhou, and W. Zhong, “An optimization strategy based on hybrid algorithm of adam and sgd,” in *MATEC Web of Conferences*, vol. 232, p. 03007, EDP Sciences, 2018.

Appendix A

Optimization of the neural network

The optimization algorithm used in this work is Adam (short for Adaptive moment estimation). The algorithm is based on an adaptive estimate of lower-order moments for first-order gradient-based optimization of a stochastic function. Here, the moment refers to the momentum term, which adds a part of the previous weight update to the current update.

This algorithm uses the running average of the gradient, momentum, and the derivatives' square. For Adam, there are three learning parameters, η, β_1 and β_2. During training, two moving averages are computed, the first moment m (mean) and the second moment ν (uncentered variance). The moving averages are retained as:

$$m_k(t+1) = \beta_1 m_k(t) + (1-\beta_1)G_k(t), \tag{A.1}$$

$$\nu_k(t+1) = \beta_2 \nu_k(t) + (1-\beta_2)[G_k(t)]^2, \tag{A.2}$$

where $G_k(t)$ stands for the gradient at moment t for a specific weight ω_k.

Defining a bias-corrected version $\hat{m}_k(t+1) = \frac{m_k(t+1)}{1-\beta_t^1}$, and $\hat{\nu}_k(t+1) = \frac{\nu_k(t+1)}{1-\beta_t^2}$, weights are updated by,

$$\omega_k = \omega_k(t) - \eta \frac{\hat{m}_k(t+1)}{\sqrt{\hat{\nu}_k(t+1)} + \epsilon}. \tag{A.3}$$

$\hat{m}_k$ and $\hat{\nu}_k$ are by definition the weighted average of the G_k and G_k^2, respectively. Adam is presented with default parameters $\beta_1 = 0.9$,$\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ [42].

Adam's optimization method improves the commonly used stochastic gradient descent algorithm (SGD) using the squared gradients to scale the learning rate. It also takes advantage of using the moving average of the gradient instead of the gradient itself, like SGD with momentum. The choice of Adam is justified in our work as it outperforms SGD in both training and validation matrices in the initial portion of the training. Still, its performance stagnates later, where SGD is better at convergence. However, a hybrid algorithm is proposed by switching from Adam to SGD at an optimal moment of the training. Such a procedure can guarantee the convergence of Adam [43].