UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE
Niels Bohr Institute

Master's thesis

Magnus Andreas Petersen Ibh

The black box learning to think outside the box

Machine learning in quantum physics

Advisors: Karsten Flensberg and Mark Rudner

Handed in: November 16, 2020

Abstract

Neural networks have shown great results in a variety of practical applications. However, most networks are applied as a black box, where no information about how the network solves the problem is known. This inhibits neural networks from being useful as a tool for scientific discoveries. The main purpose of this thesis is to study the potential of neural network driven scientific discoveries. We do that by examining different learning architectures capable of opening the black box to varying degrees. A recurrent learning architecture proposed by Zhai et al. [1] is able to develop the discrete Schrödinger equation as its update law from simulated data of a quantum particle. We recreate this learning architecture and extend it by introducing a global encoder, which enables the network model to the predict probability distributions for the ground state solutions to the discrete Schrodinger equations by storing the ground state energy as a global variable. While information about the solution method of the extended learning architecture can be fully extracted, we find that the model is unable to learn complicated systems where a non-linear update law is needed. Last, we show that a recurrent network model with a fully connected neural network as update law can learn to predict transmission probabilities of a particle scattering from potential barriers. We observe that the network needs to store 3 variables to solve the system, but ultimately, fail to identify what physical features the essential variables correspond to.

Acknowledgements

Writing a thesis during this strange year has in many ways been difficult and allowed for a lot of self-reflection. It has made it clear to me that I was only able to come this far with the help of the great people surrounding me. First, I would like to thank my two supervisors *Karsten Flensberg* and *Mark Rudner* for being open to a project outside your field of expertise and always guiding me in the right overall direction. Thank you to the people in the CMT group for providing a friendly and competent working environment. Finally, I would like to thank my friends and family for supporting me throughout the year. Especially my parents, thank you for always having my back, you are the best.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The 20th century has commonly been characterized as the "Age of Industrialization", whereas the 21st century is being denominated "Age of Information". This is with good reason, data is everywhere in our modern society. Enabled by the technical advancement in computer electronics and the exponential growth of affordable computational power and storage capacity available, states, institutions, enterprises and individuals have gathered incomprehensible amounts of data.
The ability of humans to collect and store an enormous amount of data has brought into focus the distinct difference between information and just data. We collect huge amounts of data, which has little to no meaning without a conscious and targeted effort to condense and put them into context.
The amount of work needed to manage and process a large amount of data has given a boost to the research fields known as machine learning (ML) and artificial intelligence (AI). The reason being that these concepts seem to offer a viable approach to the challenge of handling big data.
Especially artificial neural networks has seen a considerable increase in popularity and successful applications. In principle, neural networks try to emulate the key properties of the human brain. The brains decision making, learning and information storage capability is largely based on exchange of electrical signals between brain cells (so-called neurons) with different electric potential connected in a highly complicated network.
Neural networks, roughly based on this concept, form the basis of many important scientific and commercial applications. For example, convolutional neural networks [2] used for image recognition or recurrent neural networks used in natural language processing and machine translations [3]. However, just like science is only just beginning to reveal small fragments of how the brain actually works, the same can be said about most artificial neural networks. Most practical use of neural networks involves applying the networks as a black box, where no information about their solution method can be inferred.
In the recent years, the idea has been presented that the powerful, but complicated, structure of information captured in a trained "black-box" network, could be extracted on a form which would be conceivable for humans. This would potentially enable the neural networks to work as "agents" discovering and revealing underlying structures in data and observations, which science has not yet realized or been able to put into mathematical form.
The main focus of this thesis will be exploring the concept of extracting relevant information from the black box of neural network models. Specifically, the intention is to employ neural networks on simulated data of 3 relatively simple quantum systems and to extract essential information about the underlying physical system from the networks. Furthermore we want to demonstrate, what it is that makes extracting (unsupervised)[1] information from neural networks so challenging. Machine learning and AI are still very much empirical disciplines.

1.2 Application of Neural Networks in Physics

While the focus of this thesis is centered around extracting physical concepts from the black box of neural networks, practical applications in the field of condensed matter physics are, naturally also of great interest. We will here outline some of the recent results.

Neural networks have primarily shown promise in two areas of condensed matter physics, both involving many-body settings. The dimension of the Hilbert space of a many-body quantum system scales exponentially with the number of particles. As a result, computation time and memory requirements of numerical solutions often become an issue for larger systems. The poor scalability makes it favourable to attempt using neural networks where techniques addressing similar issues have already been developed. Successful application of neural network was seen in characterization of different phases of matter.

[1]The term unsupervised can sometimes be used in an ambiguous way, which we will get back to later in the thesis.

Carrasquilla and Melko apply neural networks to the square lattice ising model in [4]. A fully connected neural network, which based on the spin configuration is tasked with classifying whether the system is in the low-temperature ferromagnetic phase or the high-temperature paramagnetic phase. By training the model on configurations of different temperature they manage to provide accurate estimates on the critical temperature for the phase transition. Similarly, Nieuwenburg et al. [5] trains a fully-connected network neural model to distinguish between the topologically trivial and non-trivial phases of the Kitaev chain. Central for their work is that they first utilize a principal component analysis to group the different phases, which they then use to label the data. This makes the optimization of the network "unsupervised".
The second area, where neural networks have seen use, is in the estimation of many-body ground state wavefunctions. Here, neural models are trained to efficiently reconstruct complicated many-body ground states based on simple measurements of the system. Ref. [6–8], all utilizes a restricted Boltzmann machine (RBM) to do this. A RBM is a generative stochastic neural network model designed to learn a probability distribution over its set of inputs, which will not be considered in this thesis.

Common for all of these applications is that little attention has been paid to how the network actually solves the system.

1.3 Software and Hardware

The choice of programming platform is important, concerning AI research. Because the numerical algorithms are computationally demanding, the use of highly efficient datastructures and methods is essential.
Fortunately, several high quality, highly optimized libraries are freely available. We have chosen to build on the Python based PyTorch library provided by Facebook's AI Research lab under open source BSD license. As programming language, we use Python 3.7.
Some of the neural network models build in this thesis takes many training iterations to converge. Without enough processing power and a powerful graphics processing unit (GPU), it would be impossible to extensively test the models for different hyperparameters, which makes out an essential part of extracting relevant information. A powerful Windows 10 OS based Lenovo work station with a high-end NVIDIA GeForce RTX 2070 graphic card with 8.0 GB, has been graciously provided by the condensed matter theory group of the Niels Bohr Institute, in order to conduct this research.

1.4 Thesis Outline

The thesis consist of 3 sections, where neural network architectures are constructed to handle different quantum systems. Different neural network concepts are progressively introduced to study the prospects of network driven scientific discoveries.

- An introduction to the fundamental concepts of neural networks and machine learning is provided in section 2.
- Different neural structures are utilized to solve the 1D Schrödinger equation in 3. A recurrent network architecture proposed by Zhai et al. [1] is constructed to not only find an accurate potential-to-density mapping but also extract relevant information about the underlying physics.
- In section 4 another component is introduced to the learning architecture from the previous section which allows the machine to discover the ground state solutions to the time-independent Schrödinger equation with infinite potential boundary conditions.
- Section 5, a new recurrent network with a non-linear update law is designed to predict transmission coefficients of a particle scattering from a sequence of potential barriers. Here, we observe the challenges associated with extracting meaningful from non-linear update equations.

2 Neural Networks Overview

Before delving into the more advanced concepts of machine learning when designing networks geared towards scientific discoveries, we will first provide a brief overview of (artificial) neural network basics. Generally, neural networks can be employed to perform two distinctly different tasks, regression or classification. Classification is when the network is asked to estimate a mapping from input variables to categorical output variables. In regression, neural networks perform a mapping from input variables to continuous output variables. We will here only be concerned with regression tasks. Ref. [9] provides a detailed and intuitive description of neural networks and deep learning while [10] takes a more mathematical approach in their introduction of neural learning.

2.1 Neuron

Common (artificial) neural networks are made up of elements called neurons. A neuron, named after its biological equivalent, is the most elementary part of a neural network. A neuron maps (possibly) several inputs $\{x_1, .., x_n\}$ to an output y f_{neuron}: $\mathbb{R}^n \to \mathbb{R}$, described by the equation

$$neuron : y = \sigma\left(\sum_i^n w_i x_i + b\right) \tag{1}$$

where σ is called the activation function and $w_i \in \mathbb{R}$ and $b \in \mathbb{R}$ are adjustable parameters. These are the core elements also present in higher complexity networks. The neuron is depicted in fig.1. The green nodes contain the input which is assigned weights and transmitted to the blue node. The b parameter is independent of the input values and referred to as the bias term. At the blue node, the weighted inputs are summarized, and the sum passes through an activation function σ to determine the output y. Inspired by biological neurons, the output is sometimes called the activation of the neuron. Graphically, each arrow represents the weight that determines the importance of the node it is drawn from.

2.2 Activation Function

The activation function, σ, is an essential part of a neural network, which adds a nonlinear component to the network. A wide range of activation functions with different mathematical properties, suitable for different purposes can be used. Some of the more common and popular are introduced below.

Sigmoid: The sigmoid function

$$\sigma_{\text{sig}} = \frac{1}{1 + \mathrm{e}^{-x}}$$

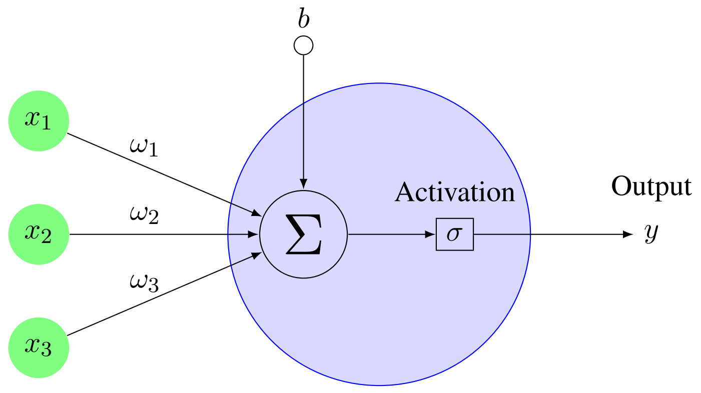


Figure 1: Graph of a single neuron.

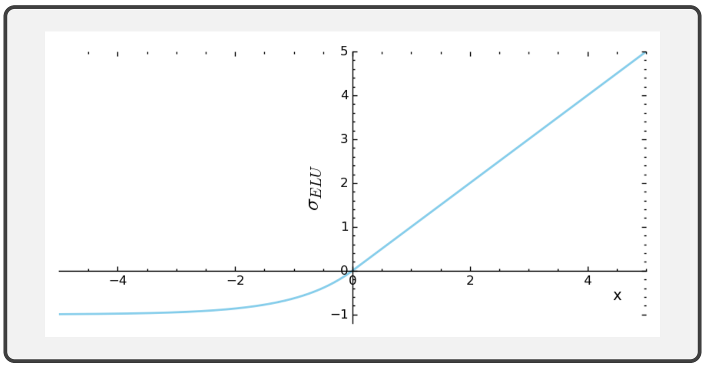


Figure 2: Elu activation with $\alpha = 1$

The sigmoid transforms input $x \in]-\infty;\infty[$ into output values between 0 and 1. For $x \gtrsim 2$ sigmoid values saturate at 1 and for $x \lesssim -2$ the $\sigma_{\text{sig}}(x)$ goes to 0 which makes the function well suited for logistic tasks and classification. The sigmoid, however, has some significant disadvantages. First, the gradient of the sigmoid function becomes zero for very high or very low values of x leading to almost no correction of the parameters. This often results in very slow or no further learning of the network. The problem is called vanishing gradients. Second, the sigmoid function is computationally expensive compared to other activation functions.

Tahn: The hyperbolic tangent activation function is defined straight forward as

$$\sigma_{\text{tanh}} = \tanh(x)$$

It is very similar to sigmoid but here the output is bound between -1 and 1, instead. Hence, this function is zero centered and handles negative output better. But it also suffers from vanishing gradients and from being computationally expensive.

ReLU: The rectified linear unit (ReLU) activation function defined as

$$\sigma_{\text{ReLU}} = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

has replaced sigmoid as the most popular activation function. There are two reasons for this. First, ReLU is very computationally efficient. Second, ReLu being a non-saturating activation function, it does not suffer from the vanishing gradients for large input values. However, ReLU activation does come with another issue called "the dead neuron problem". When inputs approach zero or are negative the gradient of the ReLU function becomes zero. This implies that With unfortunate initialization, where the majority of neurons have negative output, these neurons will have no incentive to adjust their weights and hence, the network will have limited ability to "learn".

ELU:The networks in this thesis will primarily use the exponential linear unit (ELU) activation. Introduced in Ref. [11], the activation is defined as

$$\sigma_{\text{ELU}} = \begin{cases} \alpha(\mathrm{e}^x - 1) & x \leq 0 \\ x & x > 0 \end{cases}$$

, with the parameter $\alpha > 0$. Default value is $\alpha = 1$, which will also be used throughout this thesis. The ELU activation function is shown in fig. 2 The ELU fixes many of the problems present for ReLU activation. It is continuous and differentiable at all points and unlike ReLU, it does not suffer from dying neurons since the gradient of an ELU is non-zero for all negative values. It is slightly more expensive computationally than ReLU due to the nonlinearity for $x < 0$, but still faster than sigmoid and tanh activation. Like ReLU, ELU does not suffer from vanishing gradients. Many of the pros and cons of the different activation functions will be clarified when the training process of the neural networks is outlined.

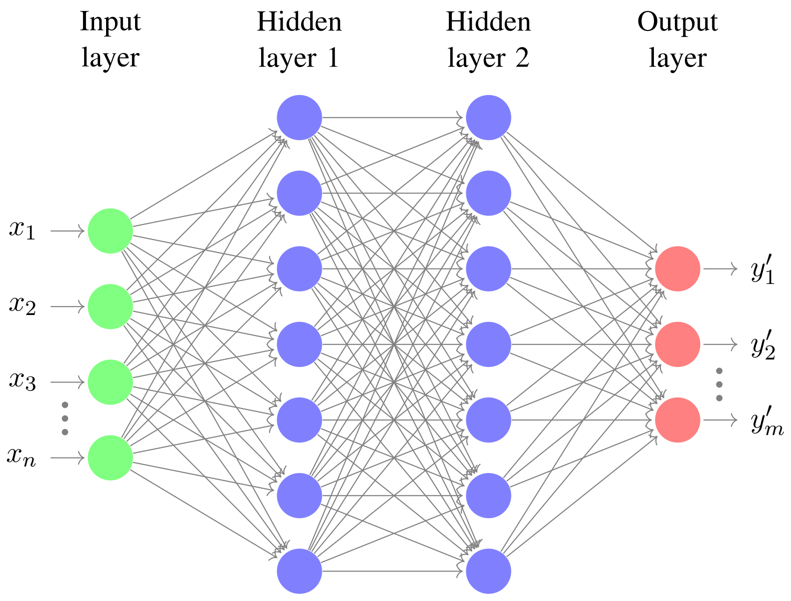


Figure 3: Illustration of a feedforward fully-connected network with an n neuron input layer, 2 hidden layers and an m neuron output layer. The activation function and number of nodes in the hidden layers are not specified.

2.3 Feedforward Fully-Connected Neural Network

The most common form of neural networks are layers of neurons put together. The neurons in a layer i receive the output from the $(i-1)^{th}$ layer and send their output to neurons in the $(i+1)^{th}$ layer. These networks are called feedforward networks because information only flows forward. If neurons in a layer receive inputs from all neurons in the previous layer, the network is referred to as a fully-connected neural network (FCNN). Even though feedforward networks need not be fully-connected we shall be using these terms interchangeably when referring to feedforward fully-connected networks. Just like for a single neuron, we can think of fully-connected networks as a function $F : \mathbb{R}^n \to \mathbb{R}^m$ that maps the input $(x_1, .., x_n)$ to the output $(y_1, .., y_m)$, see fig. 3. We are now considering a structure of numerous neuron ordered in different layers. Therefore, F can be split into a chain of simpler functions $f^{(i)}$, where $f^{(i)}$ correspond to activation/output of the neurons in the i^{th} layer.

$$F(x) = f^{(L)}(..f^{(i)}(..f^{(1)}(x)..)..),$$

with f_1 being the first layer and f_L being the last. The input of the model $x_1, .., x_n$ corresponds to the output of the individual neurons in the first layer. Hence, the first layer is called the input layer and consist of n neurons. The activation outputs from the input layer form the input to the second layer and so on. The input is sequentially propagated forward through layers until the last layer is reached. The output of the m neurons in the last layer $y_1, .., y_m$ is the output of F and $f^{(L)}$ is, hence, called the output layer. In a neural network we only know the input and the output from the final layer. The activation $f^{(i)}$ of the layers in between is not shown which is why they are called hidden or latent layers. The obscure contribution from the hidden layers, where the learning is in fact recorded, is why neural networks are often thought of as a black box. The length of the chain of functions L is associated with the depth of the model. The name "deep learning" comes from this terminology. We define the model depth as the number of hidden layers in the network, thus, not counting the input and output layer. Likewise, the number of neurons in the hidden layers is referred to as the width of the network.
Interpreting layers as many neurons/units that act in parallel, each representing a vector-to-scalar map, becomes inefficient as networks grow in size. Instead, we can think of layers as vector-to-vector functions where each element of a vector correspond to a neuron. This way the weighted sum of outputs from a previous layer can be

turned into matrix multiplication and $f^{(i)}$ can be expressed with linear algebra:

$$\begin{bmatrix} f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_{d_i}^{(i)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1}^{(i)} & w_{1,2}^{(i)} & \cdots & w_{1,d_{i-1}}^{(i)} \\ w_{2,1}^{(i)} & w_{2,2}^{(i)} & & w_{2,d_{i-1}}^{(i)} \\ \vdots & & \ddots & \vdots \\ w_{d_i,1}^{(i)} & w_{d_i,2}^{(i)} & \cdots & w_{d_i,d_{i-1}}^{(i)} \end{bmatrix} \begin{bmatrix} f_1^{(i-1)} \\ f_2^{(i-1)} \\ \vdots \\ f_{d_{i-1}}^{(i-1)} \end{bmatrix} + \begin{bmatrix} b_1^{(i)} \\ b_2^{(i)} \\ \vdots \\ b_{d_i}^{(i)} \end{bmatrix} \right) \tag{2}$$

$$f^{(i)} = \sigma \left(W^{(i)} * f^{(i-1)} + b^{(i)} \right) \tag{3}$$

$f_{i-1} \in \mathbb{R}^{d_{i-1}}$ is a d_{i-1}-dimensional vector containing the output from layer $(i-1)$, $f^{(i)} \in \mathbb{R}^{d_i}$ is a d_i-dimensional vector with the activation elements from the i^{th} layer. $W^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ is a $d_i \times d_{i-1}$ matrix that carries the tuneable weight coefficients and the vector $b^{(i)} \in \mathbb{R}^{d_i}$ has the bias terms. The symbol $*$ is used for matrix multiplication. Note that there are different weights and biases for each layer, which is indicated by a superscript of the weight matrix and bias vector labels. Last, the activation σ is applied element-wise to the $W^{(i)} * f_{i-1} + b^{(i)}$ vector. Commonly, hidden layers have the same number of neurons/nodes, thus $d = d_{i-1} = d_i$ is the size of the hidden states, specifying the width of the network.
We see that propagating from one layer to the next is given by a affine transformation (linear transformation + bias translation) and a nonlinear activation. Using this framework arbitrarily large networks can be written in the same concise format. We consider the network from fig. 3 to provide an example of a fully-connected network with 2 hidden layers and ELU activation:

FCNN:

$$F(x) = f^{(o)}(h^{(2)}(h^{(1)}(I(x))))$$

Input layer I:

$$x = (x_1, .., x_n)^T \tag{4}$$

First hidden layer $h^{(1)}$:

$$h^{(1)} = \sigma_{\text{ELU}}(W^{(1)} * x + b^{(1)}) \tag{5}$$

Second hidden layer $h^{(2)}$:

$$h^{(2)} = \sigma_{\text{ELU}}(W^{(2)} * h_1 + b^{(2)}) \tag{6}$$

Output layer $f^{(out)}$:

$$y' = f^{(out)} = W^{(\text{out})} * h_2 + b^{(\text{out})} \tag{7}$$

where y' is the output vector and network estimate of F(x)=y. The weight matrices, $W^{(1)} \in \mathbb{R}^{d \times n}$, $W^{(2)} \in \mathbb{R}^{d \times d}$ and $W^{(\text{out})} \in \mathbb{R}^{m \times d}$. Similarly, $b^{(1)} \in \mathbb{R}^d$, $b^{(2)} \in \mathbb{R}^d$ and $b^{(i)} \in \mathbb{R}^m$. Notice that non-linearity is only found in the hidden layers of the network. An activation function can be added to the input or output layer. However, in this thesis, we find it, as will be explained below, sufficient only to add non-linearity to the hidden layers of FCNNs.

2.4 Universal Approximation Theorem

One important property of FCNNs is that they are universal approximators. This implies that any continuous and bounded function can be approximated arbitrarily well (depending on network size and training) by a single hidden layer network. In Ref. [12] it was shown that a single-layer feedforward network with ReLU activation and width $n + 4$ was sufficient to approximate any continuous function of n-dimensional input variables, that is:

Theorem 1 *For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\epsilon > 0$, there exists a fully-connected ReLU network $\mathcal{A}$ with width $d_m \geq n + 4$, such that the function $F_{\mathcal{A}}$ represented by this network satisfies*

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| \, \mathrm{d}x < \epsilon.$$

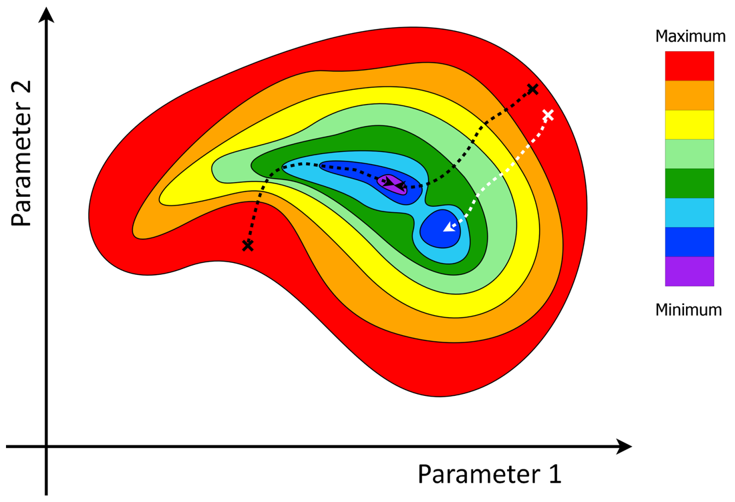


Figure 4: Sketch of gradient descent process for $C(x, \theta)$ colormap. In this case, the model has two tuneable parameters resulting in a two dimensional parameter space. The black $\times$ mark the initial values of the parameters, which result in a successful optimization to the global minimum. The white dashed line gets trapped in a local minimum.

This universality has also been shown for other activation functions, e.g sigmoid [13] and it has been demonstrated that a FCNN with any non-polynomial activation function can be universal. The reason we leave out the activation in the output layer is that FCNNs have shown to be universal even without it.

2.5 Training of Neural networks

Neural networks should be thought of as function approximation machines, trying to best estimate a given function $F(x)$. The network estimate $F_\theta(x)$ is parametrized by θ, which denotes all the tuneable weights/biases of the model $\theta = \{w^{(m)}, b^{(m)}\}$. The tuneable parameters must be adjusted for the model to best estimate $F(x)$. This procedure is known as training the network. We do this by utilizing a set of known input-outputs $\{(x, F(x))\}$. The optimal values for the parameters are chosen gradually by comparing the current output $F_\theta(x)$ to the exact target. A cost function $C(x, \theta)$ is chosen, in order to evaluate how close the prediction is to the target. The most common cost (or loss) function is the $mean\ squared\ error$ (MSE)

$$C(x,\theta)) = \frac{1}{N_{data}} \sum_{i \in data} (F(x)_i - F_\theta(\{x\}_i))^2 \tag{8}$$

also called loss function $\mathcal{L}_{MSE}$. The goal of the training is to choose parameter values such that the cost is minimized. The target data is commonly denoted $y = F(x)$ and the predicted values $y' = F_\theta(x)$.

2.5.1 Gradient Descent and Back Propagation

Finding the minimum of the cost function in the multidimensional parameter space can rarely be done analytically. Instead an iterative optimization process named gradient descent is utilized. The weights and biases of a network are usually initialized randomly[2]. The gradient w.r.t. θ, $\nabla_\theta C(x, \theta)$, is computed. Subsequently, θ is updated in the negative gradient direction

$$\theta_{t+1} = \theta_t - \tau \nabla_\theta C(x, \theta) \tag{9}$$

[2]Sometimes a certain initialization can make the training much easier and for some models a specific initialization is necessary for the model to converge.

where τ is the learning rate, a hyperparameter that dictates how much θ is updated relative to the gradient. The subscripts of θ denote the number of training iterations i.e. the number of times the parameters have been updated. The procedure is called gradient descent as we gradually, using the gradient as a guide, move toward the minimal cost. In fig. 4 a toy example shows the principle of gradient descent for a two-parameter model. Vanilla gradient descent, also known as batch gradient descent, computes the gradient of the cost function for the entire training data set. Batch gradient descent can be very slow since the gradients of the entire training set must be re-calculated in each iteration. In practice, the all training samples set is often replaced by a smaller sub-set of training samples called a mini-batch for each iteration. Then, the algorithm is called stochastic gradient descent (SGD). Stochastic, because it is assumed that the gradient of the mini-batch cost function is close to the gradient of the entire training set.
The gradients of the cost function w.r.t θ is determined with an algorithm called backpropagation. As the name suggests, the gradient is calculated starting from the cost function (output) and working backwards through the layers of the network utilizing the chain rule. Backpropagation can be made fast and efficient through matrix optimization, as is shown in [9].
All neural network libraries have efficient backpropagation algorithms implemented[3], Therefore, we will not have to worry about computing gradients 'by hand' when training a network. However, developing general equations for backpropagation and applying them to an example can greatly increase ones understanding of how neural networks are working. Moreover, calculating the gradients for a network can help outline potential issues in the training process.
We first use the chain rule to determine the gradient with respect to a single weight coefficient $w_{j,k}^{(i)}$. We define $z^{(i)} \equiv W^{(i)} * a^{(i-1)} + b^{(i)}$ as the purely linear transformation in a layer before activation is applied, and $a^{(i)} = \sigma(z^{(i)})$ as the activation output from the layer. The cost function $C(x, y, \theta)$ of an FCNN with L hidden layers is then

$$C(x, y, \theta) = C(y, f^{(out)}(a^{(L)}(..a^{(i)}(..a^{1}(f^{in}(x)))..)..)) \tag{10}$$

As a result, the gradient of the cost function for a specific weight is given by

$$\frac{\partial C}{\partial w_{j,k}^{(i)}} = \sum_{l}^{d_{out}} \sum_{m}^{d_L} \sum_{n}^{d_{L-1}} \cdots \sum_{p}^{d_{i+1}} \frac{\partial C}{\partial y'_l} \frac{\partial y'_l}{\partial a_m^{(L)}} \frac{\partial a_m^{(L)}}{\partial z_m^{(L)}} \frac{\partial z_m^{(L)}}{\partial a_n^{(L-1)}} \cdots \frac{\partial z_p^{(i+1)}}{\partial a_j^{(i)}} \frac{\partial a_j^{(i)}}{\partial z_j^{(i)}} \frac{\partial z_j^{(i)}}{\partial w_{j,k}^{(i)}} \tag{11}$$

here expressed in a non-vectorized form, where $a_m^{(L)}$ is the activation of the neuron m in the L$^{\text{th}}$ hidden layer etc. The calculation quickly becomes quite an index heavy task, even with just a modest number of layers and neurons. But in fact, we only need 4 equations to find the gradient with respect to the parameters of an FCNN of any given depth and width.
First, the partial derivative w.r.t. the output layer $\frac{\partial C}{\partial y'_l}$ in vector notation becomes

$$\nabla_{y'} C = (\frac{\partial C}{\partial y'_1}, .., \frac{\partial C}{\partial y'_l}, .., \frac{\partial C}{\partial y'_N})^T \tag{12}$$

for quadratic cost $\nabla_{y'} C = 2(y' - y)$.
Second, $\frac{\partial a_m^{(i)}}{\partial z_m^{(i)}} = \frac{\partial \sigma(z_m^{(i)})}{\partial z_m^{(i)}} \equiv \sigma'(z_m^{(i)})$ is the derivative of the activation function. Since this is an element-wise activation, it is trivially generalized to $\frac{\partial a^{(i)}}{\partial z^{(i)}} \equiv \sigma'(z^{(i)})$. Additionally, from considering a single neuron, it follows that $\frac{\partial z_n^{(i)}}{\partial a_m^{(i-1)}} = w_{n,m}^{(i)}$. Then moving back a layer with the chain rule is given by:

$$\frac{\partial C}{\partial z_n^{(i)}} = \sum_m \frac{\partial C}{\partial z_m^{(i+1)}} \frac{\partial z_m^{(i+1)}}{\partial a_n^{(i)}} \frac{\partial a_n^{(i)}}{\partial z_n^{(i)}} = \sum_m \frac{\partial C}{\partial z_m^{(i+1)}} w_{m,n}^{(i+1)} \sigma(z_n^{(i)})$$

[3]including Pytorch, used in the thesis

, which in matrix form corresponds to

$$\nabla_{z^{(i)}} C = \sigma'(z^{(i)}) \odot \left(\left(W^{(i+1)} \right)^T * \nabla_{z^{(i+1)}} C \right) \tag{13}$$

where $\odot$ denote the elementwise multiplication and $*$ matrix multiplication. $\frac{\partial z^{(i+1)}}{\partial a^{(i)}} = \left(W^{(i+1)} \right)^T$ is a matrix whose components are the partial derivatives $\frac{\partial z_n^{(i)}}{\partial a_m^{(i-1)}} = w_{n,m}^{(i)}$, where the numerator index refer to the column placement and denominator index the row.
Third, we note that $\frac{\partial z_j^{(i)}}{\partial w_{j,k}^{(i)}} = a_k^{(i-1)}$. The partial derivative only returns a non-zero value if the weight is present in the node $z^{(i)}$. This can also be written on matrix form as an outer product.

$$\frac{\partial C}{\partial W^{(i)}} = \nabla_{z^{(i)}} C \frac{\partial z^{(i)}}{\partial W^{(i)}} = \begin{bmatrix} \frac{\partial C}{\partial z_1^{(i)}} \frac{\partial z_1^{(i)}}{\partial w_{1,1}^{(i)}} & \cdots & \frac{\partial C}{\partial z_1^{(i)}} \frac{\partial z_1^{(i)}}{\partial w_{1,d_{i-1}}^{(i)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial z_{d_i}^{(i)}} \frac{\partial z_{d_i}^{(i)}}{\partial w_{d_{i-1},d_1}^{(i)}} & \cdots & \frac{\partial C}{\partial z_{d_i}^{(i)}} \frac{\partial z_{d_i}^{(i)}}{\partial w_{d_i,d_{i-1}}^{(i)}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial C}{\partial z_1^{(i)}} a_1^{(i-1)} & \cdots & \frac{\partial C}{\partial z_1^{(i)}} a_{d_{i-1}}^{(i-1)} \\ \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial z_{d_i}^{(i)}} a_1^{(i-1)} & \cdots & \frac{\partial C}{\partial z_{d_i}^{(i)}} a_{d_{i-1}}^{(i-1)} \end{bmatrix} = \left(a^{(i-1)} \otimes \nabla_{z^{(i)}} C \right)^T \tag{14}$$

Fourth, when finding the gradient w.r.t. the bias we have $\frac{\partial z_j^{(i)}}{\partial b_j^{(i)}} = 1$ which can also be on a vectorized form as

$$\frac{\partial C}{\partial b^{(i)}} = \nabla_{z^{(i)}} C \frac{\partial z^{(i)}}{\partial b^{(i)}} = \nabla_{z^{(i)}} C \tag{15}$$

Using Eq.(12)-(15) we see that Eq.(11) becomes

$$\frac{\partial C}{\partial w_{j,k}^{(i)}} = a_k^{(i-1)} \sigma(z_j^{(i)}) \odot \left[(\sum_p \frac{\partial z_p^{(i+1)}}{\partial a_j^{(i)}}) * \left[\cdots \sigma(z^{(L-1)}) \odot \left(\left(W^{(L)} \right)^T * \left(\sigma(z^{(L)}) \odot \left(\left(W^{(out)} \right)^T * \nabla_{y'} C \right) \right) \right) \right] \right] \tag{16}$$

and for the entire weight matrix of a layer we have the gradient matrix.

$$\frac{\partial C}{\partial W^{(i)}} = \left(a^{(i-1)} \otimes \left(\sigma(z^{(i)}) \odot \left[\left(W^{(i+1)} \right)^T * \left[\cdots \sigma(z^{(L-1)}) \odot \left(\left(W^{(L)} \right)^T * \left(\sigma(z^{(L)}) \odot \left(\left(W^{(out)} \right)^T * \nabla_{y'} C \right) \right) \right) \right] \right] \right) \right)^T \tag{17}$$

The order of the operations is important, starting with the cost function we work our way backwards through the network [4].
The gradient w.r.t. a bias vector only differ in the final step.

$$\frac{\partial C}{\partial b^{(i)}} = \sigma(z^{(i)}) \odot \left[\left(W^{(i+1)} \right)^T * \left[\cdots \sigma(z^{(L-1)}) \odot \left(\left(W^{(L)} \right)^T * \left(\sigma(z^{(L)}) \odot \left(\left(W^{(out)} \right)^T * \nabla_{y'} C \right) \right) \right) \right] \right] \tag{18}$$

Eq.(12)-(15) provide us with an efficient method to calculate the gradient of the cost function for an FCNN. When dealing with other network types, some steps in the backpropagation procedure will have to be modified. but in principle it works the same.
When networks become deeper, we keep multiplying weight coefficients. This gives rise to certain training issues called exploding and vanishing gradients. If the majority of the weights are greater than 1 the gradient

[4]As a convention the gradients are propagated backwards as column vectors. However, if we had used row vectors applied from the other side we would not have to transpose the matrices. But we have kept convention of with the convention set up in [9]. It demonstrates how finding the gradients is the reverse process of feedforward output generation.

quickly blow up and, similarly, for weights smaller than 1 the gradient will be 0. This is especially challenging for Recurrent Neural Networks (RNN) where the same weights are used repeatedly[5]. The derivative of the activation function also contributes to the gradient, explaining why asymptotic activation functions, like sigmoid, are problematic as the derivative of the activation goes to 0 for $z_m^{(i)} \lesssim -2$ or $z_m^{(i)} \gtrsim 2$ leading to the gradients becoming 0.

2.5.2 Optimizers

When dealing with non-convex cost (or loss) surfaces, standard SGD gradient descent optimization will often fail to discover the optimal parameter values, e.g. get trapped in local minima[6]. This is illustrated in fig. 4, where the white path represent initial values for the parameters that ultimately get trapped in a local minimum. In order to combat difficult loss surfaces other optimizers, utilizing a slightly more advanced optimization routine, has been developed. The Adaptive Moment Estimation, or Adam optimizer, [14] will be the primary optimizer implemented in models for this thesis. Adam combines the advantages of two other optimizers, AdaGrad [15] and RMSprop [16]. Similar to RMSprop and AdaGrad, Adam utilizes an exponentially decaying average of previous squared gradients v_t, but in addition it takes the exponentially decaying average of the previous gradients m_t into account as well. The averages m_t and v_t are given by:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{19}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{20}$$

β_1,β_2 are hyperparameters, that set the exponential decay rates and g_t denotes the gradient. The averages correspond to estimates of the first and second[7] moment of the gradients. Adam differs from RMSprop in that it has a bias correction:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{21}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{22}$$

which is of significant importance for the early training iterations. The Adam update rule for a parameter θ is:

$$\theta_{t+1} = \theta_t - \frac{\tau}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \tag{23}$$

where τ is the initial stepsize/learning rate and ϵ is a term that prevents division by zero. This update rule makes Adam computationally efficient and assures fast convergence.
It should be mentioned that higher-order optimization algorithms, such as second-order hessian free optimization [17], exist. However, the higher-order optimization methods have yet to gain popularity. The potential information gained from the second-order derivatives does not compensate for the increased computation time, especially for high dimensional parameter spaces. It has also yet to be demonstrated, if higher-order optimizers can converge in situations, where first-order optimizers fail.

2.5.3 Overfitting and Underfitting

Finding the minimal cost is the assignment given to the learning algorithm. If we want the network to prioritize specific solutions, modifications can be made to the cost function. One type of modifications are known as regularization where terms are added, that forces (or regularizes) small coefficient estimates towards zero. This technique is an analog to Occam's razor that encourages the simplest possible solution to the model. Practically, regularization is mainly used as a tool to prevent overfitting. Overfitting is when the network picks up on the

[5]RNNs will be introduced in a later section.

[6]Actually, local minima are not the biggest issue since it is unlikely for the gradient w.r.t. to all parameters to be zero

[7](uncentered/biased variance)

statistical variation of the data as if it was part of the underlying function we are trying to model. As a result, an overfitted model contains more variability than can be justified by the data. On the other hand, underfitting can occur if a network has not been provided enough degrees of freedom (depth,width,non-linearity) to adequately capture the underlying system. In both cases, the resulting network models can give accurate predictions when applied to input data, that they were trained on, but potentially fail to generalize the predictions for input data not specifically used in the training.

2.5.4 Sample Size

The amount of data needed to successfully train a neural network depends on various factors. Generally, the more data the better, since with a sufficiently large dataset overfitting can be avoided. Machine learning is a process of induction, i.e. the model can only learn from the data it is subjected to. Hence, if input data is not diverse or populated enough, there is a high probability that the true underlying features will not be captured by the network. Instead, the network will be able only to accurately predict the narrow/small dataset it has been trained on.
However, the training set can also be too large[8]. In these situations the training time can be unnecessarily long. One needs to find the right balance, where the data amount is large enough to statistically represent the underlying structures and features without inflicting unnecessary long training times.
Eventually, a pragmatic heuristic approach works the best. Pick a reasonable sample size, based on the size and complexity of the model. Divide the sample set into training and validation sets by random selection and train the network. Then analyse the network models ability to predict the validation set by looking for signs of overfitting, underfitting and poor accuracy. Possible solutions to issues of that kind, is to increase variance/size of the training data set or to adjust hyperparamerers (see next section). If the model appears healthy, a smaller training data set size can be attempted to reduce the training time if needed.

2.6 Hyperparameters

It should be apparent by now that a multitude of different parameters can affect the training and performance when modelling neural networks. The parameters that are not optimized by the learning algorithm, but rather selected by the network designer, is called hyperparameters. We here list the relevant hyperparameters, for this thesis, in the respective part of the model construction they belong to along with the default values used throughout the thesis.

- Training:
  - Epoch : Number of training iterations/steps, no default number assigned, as the requirement is highly dependent on the model.
  - Batch size: Random number of samples drawn from the training set used for one training iteration, default batch size is 50.
  - Weight initialization: The model parameters are by default given initial values between 0 and 1 drawn from a uniform distribution.(i.e. random values)
- Adam Optimizer:
  - β_1: Control the exponential decay rates for the exponential gradient average, default value $\beta_1 = 0.9$.
  - β_2: Control the exponential decay rates for the exponential squared gradient average, default value $\beta_2 = 0.999$.
  - ϵ: Term in parameter update rule that prevent division by 0, default value $\epsilon = 10^{-8}$.
  - τ: Initial learning rate, default value $\tau = 10^{-4}$

[8]Especially with simulated data where there is only practical limits to the number of samples one can produce

- ELU Activation:
  - α: Lower bound in ELU activation function is $-\alpha$, default $\alpha = 1$.
- Model size:
  - Width: Number of neurons in the hidden layers, no default value.
  - Depth: Number of hidden layers in the network[9] , no default value.

[9]When modelling more advanced architectures such as recurrent networks, depth refer to the stacking of recurrent cells.

3 Schrödinger Machine

In this section, we study a simplified version of the single particle Schrödinger equation (SE) first investigated by Zhai et al. in [1]. In their paper, they construct a learning architecture, called the *Schrödinger Machine*, which can accurately find the probability density ρ_i given the potential V as input. Additionally, the network architecture is able to filter out redundant information such that the Schrodinger equation and wavefunction ψ_i emerge. We reproduce their results for validation purposes, but also further investigate the discovered solution and the capabilities of the learning architecture.

3.1 Physical System

The data, we expose the neural network architecture to, is generated by solving the dimensionless time-independent Schrödinger equation in 1D where the energy is absorbed into the potential giving the equation.

$$V(x)\psi(x) = \partial_x^2\psi(x) \tag{24}$$

x being the particles position, $V(x)$ is the potential, $\psi(x)$ is the wavefunction and ∂_x is shorthand for the spatial derivative w.r.t. x. In this version $\frac{\hbar}{2m} = 1$. Analytical solutions can be obtained by looking at intervals of constant potential. $V(x)$ is discretized into a sequence of numbers $\{V(x_i) = V_i\}$ such that $V(x) = V_i \quad x \in [x_i; x_{i+1}[$. The constant a is introduced as the width of each potential pillar such that $x_i = ia$. The physics is covered by literature e.g. [18]. We are in this case considering freely moving particles i.e. the energy of the particle being higher than the potential for all x. Since $V(x)$ is taken relative to the energy E, this is effectively setting $E = 0$ and $V_i < 0$. We define $k_i = \sqrt{-V_i}$ such that general solutions to eq.(24) take the form

$$\psi(x) = A_i \sin(k_i x) + B_i \cos(k_i x), \quad x \in [x_i; x_{i+1}[\tag{25}$$

with the spacial derivative

$$\partial_x\psi(x) = k_i A_i \cos(k_i x) - k_i B_i \sin(k_i x), \quad x \in [x_i; x_{i+1}[\tag{26}$$

The entire solution $\psi(x)$ is therefore a piecewise function made up of eq.(25) as building block for each piece of constant potential. The concept is illustrated in fig. 5. Note that because $V_i < 0$, A_i and B_i will always be real resulting in $\psi_i \in \mathbb{R}$[10].
The constants A_i and B_i can be determined by matching $\psi(x)$ and $\partial_x\psi(x)$ at the x_{i+1} boundary, imposing that $\psi(x)$ and $\partial_x\psi(x)$ are continuous for all x (not considering infinite potentials). Solving for A_{i+1} and B_{i+1} results in the following relations[11]:

$$\begin{aligned} A_{i+1}k_{i+1} = k_i &\Big(A_i \cos(k_i x_{i+1})\cos(k_{i+1}x_{i+1}) - B_i \sin(k_i x_{i+1})\cos(k_{i+1}x_{i+1}) \Big) \\ +k_{i+1} &\Big(A_i \sin(k_i x_{i+1})\sin(k_{i+1}x_{i+1}) + B_i \cos(k_i x_{i+1})\sin(k_{i+1}x_{i+1}) \Big) \end{aligned} \tag{27}$$

$$\begin{aligned} B_{i+1}k_{i+1} = k_i &\Big(B_i \sin(k_i x_{i+1})\sin(k_{i+1}x_{i+1}) - A_i \cos(k_i x_{i+1})\sin(k_{i+1}x_{i+1}) \Big) \\ +k_{i+1} &\Big(B_i \cos(k_i x_{i+1})\cos(k_{i+1}x_{i+1}) + A_i \sin(k_i x_{i+1})\cos(k_{i+1}x_{i+1}) \Big) \end{aligned} \tag{28}$$

The two recurrence relations allows us to find all $\{A_i, B_i\}$ as long as the first A and B are provided. The

[10] This is a different argument from the fact that stationary eigenstates of the time-independent SE can be chosen as real without loss of generality. That is because the the time-independent SE is a real second order differential equation.

[11] See Appendix A for the derivation

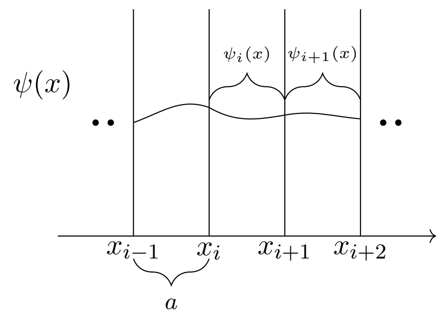


Figure 5: Illustration of the wavefunction as a function of position $x_i = ia$. $\psi(x) \in [x_i; x_{i+1}[$ is denoted $\psi_i(x)$ for clarity.

choice of A_0 and B_0 is arbitrary, but we will in this thesis use the fixed initial condition $A_0 = B_0 = 1$. These extended-state solutions are naturally not normalizeable. Thus, the probability density profile $\rho(x)$ that arises is not a real probability, since it is not normalized. Even though the states are not normalized, they are useful as a proof-of-concept study to show the power of the neural network architectures. We shall still refer to $|\psi(x)|^2$ as a probability density profile $\rho(x)$ and, at a discrete point, as the (probability) density $|\psi(x_i)|^2 = \rho_i$.

3.1.1 Data generation

The discrete data fed to the learning architectures is generated the following way: The sequence $\{V_i\}$ is randomly generated between the lower bound ,V_{min} and 0. Starting from fixed initial conditions A_0 and B_0 we use eq. (27) and (28) to construct $\psi(x)$ as in eq.(25). Finally, the probability density at x_i is $\psi_i^2 = \rho_i$. The data generation process can be summed up in the following steps [12]:

- Fixed V_0 and the rest of the sequence is $V_i = V_{\text{min}} \times r_i - R$, where V_{min} is the lower bound on the potential, r_i is a number between 0 and 1 drawn from a uniform distribution for each V_i. Likewise, R is a random number uniformly distributed in $[0, 1]$, but R is the same for each V-sequence to simulate different energies.
- Smoothing of potential sequences by applying the relation $V_{i+1} = \frac{1}{2}(V_i + V_{i+1})$, on each sequence a random number of times, between 0 and q ($q \in \mathbb{N}$). Note this step is not strictly necessary, but useful to generate a diverse dataset.
- Starting from fixed initial condition A_0 and B_0, use from eq. (27) and (28) to get $\psi(x_i)$ and probability density $\psi_i^2 = \rho_i$.

Values of the constants needed in the data generation are shown in table 1. Training and testing have also been done with different values, but the ones listed are the values used for the results in the thesis, unless explicitly stated otherwise.

Constants	N_L	a	A_0	B_0	V_{min}	V_0	q
	400	0.1	1	1	-2	-1	20

Table 1: Parameters/Constants used in the data generation.

Following the approach from section 2.5.4, we create 15000 different V_i and ρ_i sequences with $N_L = 400$ elements in each sequence, 10000 used for training and 5000 used as validation. The validation data is excluded from the training process and can therefore be used to identify issues like over- and underfitting in the training of neural models.

[12]code can be found in appendix A

3.2 Fully-connected network

The simplest form of a neural network, as described in chapter 2, is the feedforward fully-connected neural network. It is educational to first apply a FCNN model. It shows the predictive power of even simple neural network structures, but also highlights how they are ill-suited for the discovery of new scientific concepts.
The network has to perform a potential-to-density mapping $f : \mathbb{R}^{N_L} \to \mathbb{R}^{N_L}$, transforming a sequence of $N_L = 400$, discrete potential values into the corresponding ρ_i which means the input vector and output vector will both have the dimension N_L. The network consists of an input layer, a number of hidden layers and an output layer. In the following, we shall consider FCNN models with 1, 2 and 3 hidden layers. The structure of a FCNN model with 2 hidden layers is shown in fig. 3 but with the input being the potential sequence $V = \{V_i\}$ and output the density sequence $\{\rho_i\}$. All models have a width of $N_{\text{hid}} = 500$ nodes in the hidden layers. The models are evaluated with the MSE loss function, $\mathcal{L}_{MSE}$, where

$$\mathcal{L}^{\rho_i}_{MSE} = \frac{1}{N_{\text{total}}} \sum_{i \in \text{sequence+batch size}} (\rho_i - \rho'_i)^2 \tag{29}$$

with ρ_i being the simulated data and ρ'_i being the output of the network models. The models are trained with the Adam optimizer with a learning rate, $\tau = 1 \cdot 10^{-4}$.
The loss of the training and validation data over the training period is shown in fig. 6. The figure shows the 3 different FCNN models trained for 50000 iterations all with a entire batch of samples. The combination of the training and validation loss must be considered to evaluate the performance of each model. The validation loss gives an unbiased estimate of the prediction accuracy of the different models. Thus by comparing training and validation loss we can deduce, if the models have learned the underlying physical features of the data or just minimized the loss by overfitting or underfitting to the given training set. The evaluation is complicated in that symptoms of over- and underfitted models can be quite similar, when examining the training history of a network.
The training history of the model with 1 hidden layer show signs of either over- or underfitting. Here, the loss of the training set continually decrease during the training period whereas the loss from the validation starts to increase again after a certain number of iterations. We train the models on a simulated dataset, where no statistical uncertainty is added to ρ_i. It is, therefore, not possible for the models pick up on statistical fluctuations, which means that the 1-layer model must be underfitted.
Underfitting is primarily caused by two issues. First option is that the network is not large enough to develop an adequate solution to the Schrödinger equation and thus can only try to limit the loss of the training set with an imperfect solution. Second option is that underfitting can occur when the value set of the training data does not span the entire value set of the underlying system. In that case, a simpler model than actually needed can possibly reduce the training loss. But said model will fail when exposed to input data outside the scope of the training set.
To uncover these issues, we compare the 1-layer model to the two multilayer models. The models with 2 and 3 layers show a much smaller discrepancy between training and validation loss. The difference is smallest for the 3-layer model. This implies that the 1-layer network with 500 neurons did not have enough expressive power to perform the mapping. Hence, the current training dataset does not have to be expanded since underfitting does not seem to occur for the models with 2 and 3 hidden layers.
The performance of the models can be visualized by comparing the predicted values to the generated validation tagets. The predictions of the three models for one test sample is shown in fig. 7. One sample is obviously not enough to gauge the capability of the models. As shown, looking at $\mathcal{L}^{\rho_i}_{MSE}$ for test sample sets, is one way to evaluate the performance of the models. However, this metric does not tell us anything about the relative accuracy of the model. Hence, additionally, we will want to look at the average relative error $\langle \Delta \rangle$ over the entire validation set with

$$\langle \Delta \rangle = \frac{1}{N_{\text{data}} N_{\text{in}}} \sum_i \frac{|\rho'_i - \rho_i|}{\rho_i} \tag{30}$$

N_{data} and N_{in} being the number of samples and sequence length, respectively. It should be noted, that this assessment method will have an issue with systems where target values are close to zero. Such values can result in divergent relative errors, which does not reflect an accurate evaluation of the performance of the network. We mitigate this effect by not evaluating target data within the first quartile of values, i.e. disregarding targets in the neighbourhood of zero. This approach obviously involves a risk of bias, but we argue that since the loss does not have any inherent preference towards specific ρ_i values we still get a valid estimate of the relative error this way without having the divergence problem. This is true because the network is trained to minimize absolute, not relative error.
The most effective way to evaluate the quality of the models is to compare test loss and relative error $\langle\Delta\rangle$ for the different models. Here we find:

$$\langle\Delta\rangle_1 = 41\% \quad , \quad \langle\Delta\rangle_2 = 11\% \quad , \quad \langle\Delta\rangle_3 = 8\%$$

$\langle\Delta\rangle_n$ is the relative uncertainty of the n-hidden layer network. We see that the accuracy of 2-layer and 3-layer model is significantly better than the 1-layer.

We also have to consider the the number of trainable parameters in the models. The number of parameters for a FCNN model with depth n is given by

$$N_p(\text{n}) = (N_{\text{in}} + 1)N_{\text{hid}} + (n-1)(N_{\text{hid}} + 1)N_{\text{hid}} + (N_{\text{hid}} + 1)N_{\text{out}} \tag{31}$$

Where N_p is the number of trainable parameters, n is the number of hidden layers and N_{in}, N_{hid}, N_{out} are the dimensions of the input, hidden and output layer, respectively. The number of parameters for each of the 3 models is hence:

$$N_p(1) = 400900 \quad , \quad N_p(2) = 651400 \quad , \quad N_p(3) = 901900$$

All practical FCNN models have a significant number of parameters and increasing the size of a models naturally leads to longer training times. Fortunately this issue can, with current computer technology, readily be dealt with by simply providing enough parallel hardware processing power in the form of computer clusters or custom made processing units available at reasonable cost.
Particularly for FCNNs, utilizing GPU (i.e. graphic card) based systems to train the network is a competetive option. Performing training on GPU's is really efficient for FCNNs because the procedure consist of many large matrix operations for which these floating point optimized processors excel. When the neural network architecture becomes more intricate the benefit of using GPU's compared to general CPU's is, however, significantly reduced.

The key problem, however, with an FCNN as a tool for discovering and extracting relevant parameters from a physical system is that the network is applied as a black box where relevant and redundant information gets mixed up in the different layers. Looking at the FCNNs above we can basically extract zero information about the physical system from the individual values of the hidden nodes. FCNNs applied this way can only be used in a supervised fashion and cannot aid in unsupervised discoveries. There exist several alternative network architectures to address this problem, some of which we will explore in the following sections.

Conclusion

FCNN Models with 2 and 3 hidden layers perform reasonably well and have a quite low relative error. The biggest obstacle is the high number of parameters in the model. This feature makes it impossible to extract any meaningful information from the hidden layers.

3.3 Recurrent Neural Network

We have until now only applied fully connected networks, where each neuron is connected to all the neurons is the previous and next layer. The thousands of trainable weights mean that the individual weight holds no

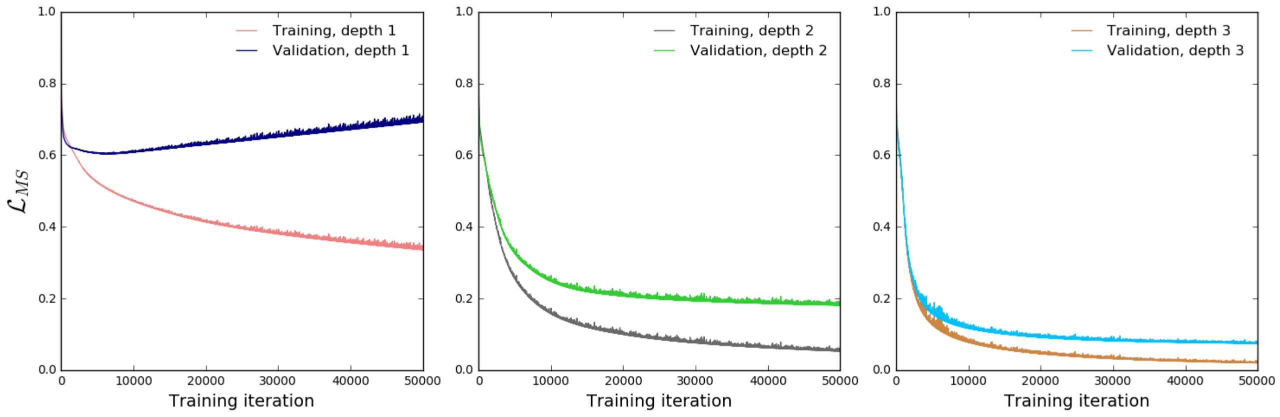


Figure 6: Loss for 50000 training iterations with different depth of the fully-connected neural network. Training set: 10000 samples. Validation set 5000 samples. (left) Underfitted model that is not able to 'learn' the proper mapping. (middle) and (right) Shows reduced loss for both validation and training data, The networks can predict the probability density reasonably well.

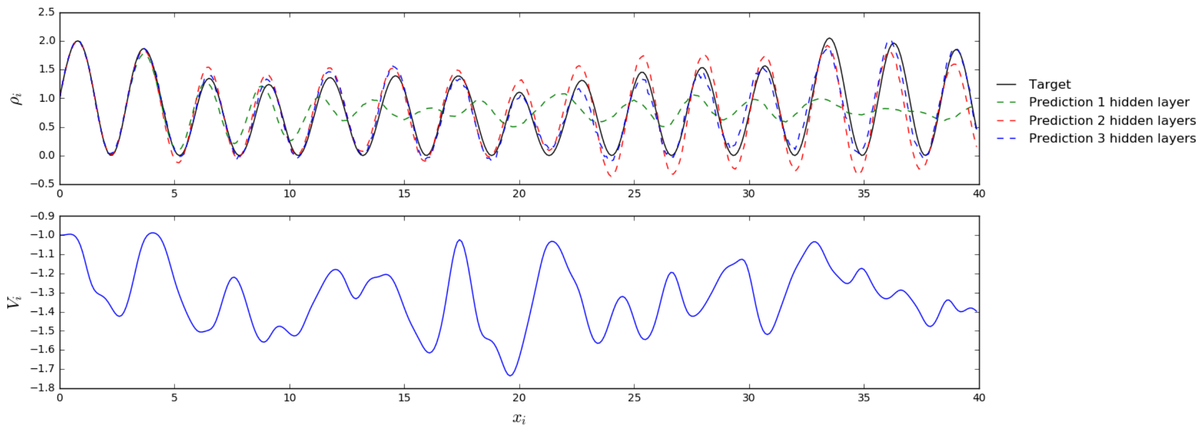


Figure 7: Prediction accuracy of fully-connected networks with different depth. Input potential V_i shown in lower graph

meaningful or directly interpretable information about the system the model was trained on. One way to address this issue is to apply a network architecture where far fewer parameters are needed. This is the case for the $recurrent\ neural\ network$ (RNN). Recurrent networks differ from fully-connected in how the input data is handled by the model.

Standard feed-forward neural networks have a fixed amount of input data which is computed all at once providing a fixed output size as well. RNNs instead has a sequence of inputs, where input is handled in steps one at a time. The unique feature of RNNs is that they, in most cases, consist of a single RNN cell. The RNN cell takes an input from the sequence and a hidden state from previous RNN cell. The same cell is used for all inputs, which potentially can help us understand how the architecture adapts to the problem at hand.

A series of calculations is done in the cell at each step (i) before producing an output. After the calculations, part of the output, called the hidden state h_i, is passed on to the next cell. The process is repeated until the sequence of inputs is completed. The structure of an RNN is illustrated in fig. 8. The non-linear transformation performed on the previous hidden state to obtain the new and updated hidden state is the $update\ law$ of the network. The hidden state h_i is a d-dimensional vector, $h_i \in \mathbb{R}^d$. Thus, d is the number of hidden variables passed on in each iteration. The RNN structure also introduces a new hyperparameter in the form of the initial hidden state. In this entire section (3), the initial hidden state $h_0 \in \mathbb{R}^d$ is initialized as an all-ones vector. The reason why this is

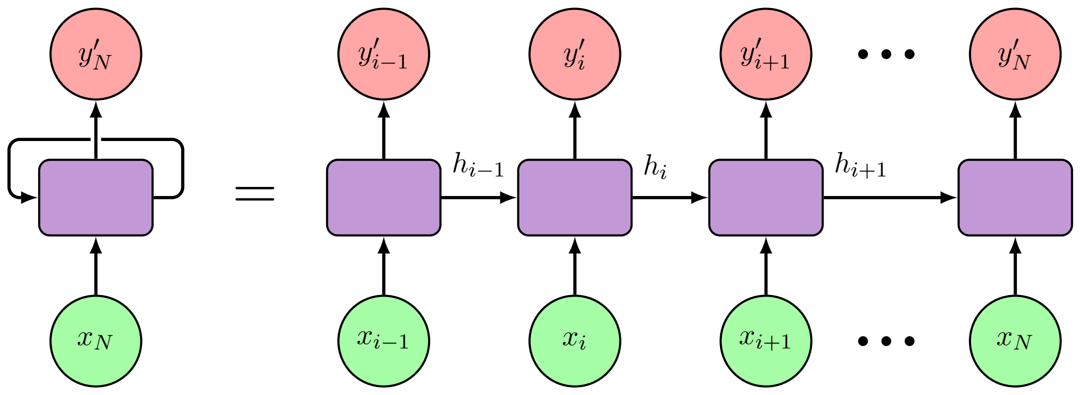


Figure 8: Global RNN structure of a sequence-to-sequence mapping. A hidden state h_i is fed back into the cell in each step of the sequence. In the cell (purple box), the hidden state is updated, based on the input x_i, and the output y_i is produced.

working will be addressed later in section 3.5.
There is a great deal of freedom in the choice of RNN cell. The standard cell used by many neural network libraries is shown in fig. 9a.
In this cell, the previous hidden state h_{i-1} and input x_i [13] are concatenated and then subjected to the update law (linear transformation + activation) to find the updated hidden state h_i. The updated hidden state is then duplicated and one copy passed on to the next step in the sequence. The other copy passes through an output gate producing the i^{th} output y'_i of the model. The update law and output gate for a standard cell is given by the following equations:

Standard update law:

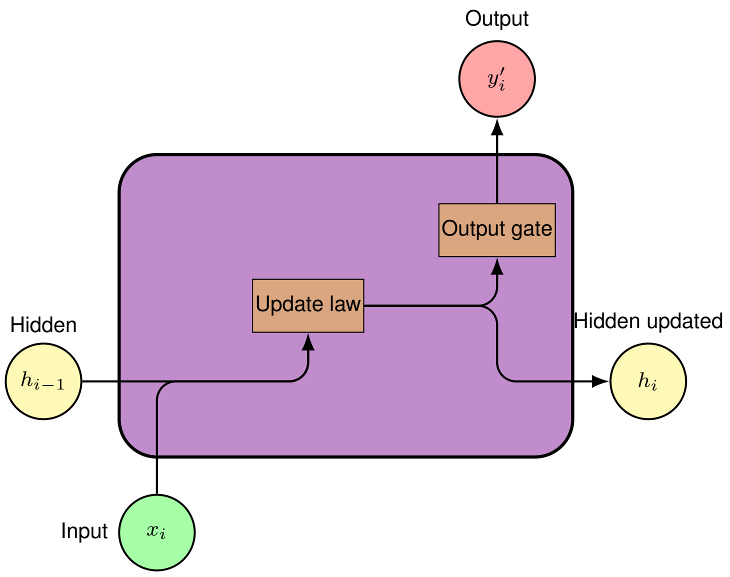
$$h_i = \tanh(W^{(h)} * [h_{i-1}, x_i] + b^{(h)}) \tag{32}$$

FCNN-based output gate:

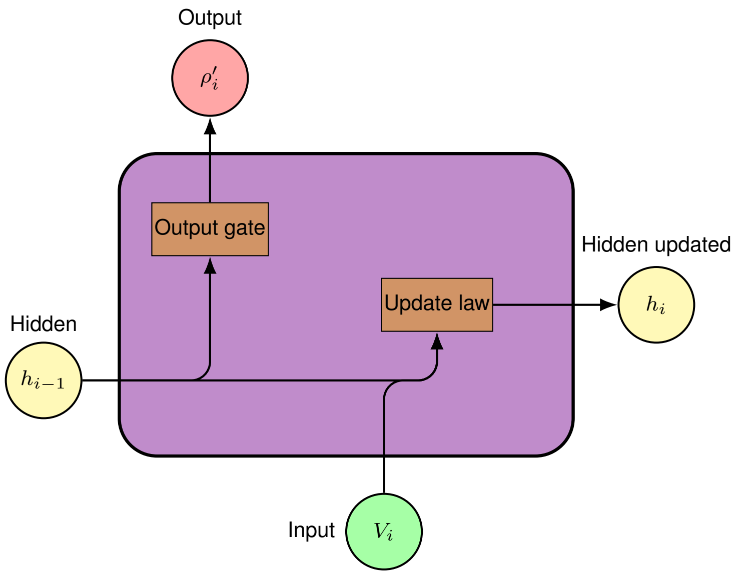
$$y'_i = W^{(\text{out})} * \text{ELU}(W^{(h_1)} * h_i + b^{(h_1)}) + b^{(\text{out})} \tag{33}$$

The first equation is the update law, where $[h_{i-1}, x_i] \in \mathbb{R}^{I+d}$ is the concatenated hidden/input vector. $W^{(h)}$ is a $(I + d) \times d$ weight matrix and $b^{(h)}$ is the bias vector. I is the dimension of the input at each step.
The second equation (33) defines the output gate function. The choice of output gate is flexible and depends on the problem, above a 1 layer FCNN is listed. Note that the cell utilizes similar operations as the regular feed-forward networks except they only work on one input in a sequence at a time.
Since the hidden state can store information about previous inputs, it functions as a memory. Being able to utilize information from previous inputs make RNN's very effective in handling problems like Natural Language Processing and machine translation, see [3] for a review of these topics.
Aside from the benefit of having fewer parameters, there is also a physically sound reason for applying an RNN model to the Schrödinger equation set of physical systems. Doing some basic examination of the data one discovers that only local input seem to be relevant for the mapping between potential and probability density. Therefore, the use of RNN models is justified since having all inputs connected is not needed. In a later section, we will study a case with infinite-well boundary conditions where this does not apply.
This realization motivates us to make some slight adjustment to the recurrent cell. For the quantum system, we chose the output gate to be a scalar product of a $1 \times d$ projection vector $W^{(p)}$ and h_{i-1}. Thus, the output ρ'_i is a linear projection of the **previous** hidden state h_{i-1}. We want relevant information stored in the hidden states. Hence, by having the output as a weighted sum of the hidden nodes, the density/output ρ_i must be contained within the h_{i-1} and properly updated each step. This is graphically illustrated in fig. 9b. At first it might seem

[13]Here the x_i-vector is general notation for input and not referring to the spacial coordinate of the physical system



(a) Standard RNN cell as implemented in most NN libraries. The concatenated $[h_{i-1}, x_i]$ vector is transformed into the updated hidden state by the RNN update equation. The output gate takes the updated hidden state h_i as input to produce the output.



+

(b) Alternative RNN cell. The projection gate generates the output from the previous hidden state. Subsequently the concatenated $[h_{i-1}, V_i]$ vector is turned into the updated hidden state h_i by the RNN update law.

Figure 9: Composition of the RNN cells. The merging of two arrows signals a concatenation of vectors.

illogical to project the output before the hidden state is updated. However, this simply forces the network to store ρ_i in the hidden state before the update law is applied rather than after. That way information is always propagated forward. Based on the current state ρ_i, V_i etc. the update rules estimate how ρ_i is affected by this. The projection matrix also potentially simplifies the training process since fewer parameters are involved. Granted, if the update law equation is not capable of mapping ρ_{i-1} to ρ_i, the projection gate will be of little help. As a first attempt we construct a RNN with the cell.

Update law:

$$h_i = \tanh(W^{(h)} * [h_{i-1}, V_i] + b^{(h)}) \tag{34}$$

Output gate/ Projection:

$$\rho_i' = W^{(p)} * h_{i-1} \tag{35}$$

The input $V_i \in \mathbb{R}$ and the output $\rho_i' \in \mathbb{R}$ are scalars. An advantage of the RNN architecture is the scalability of the model. The choice of sequence length is arbitrary as the single cell of parameters is reused for each input. If the trained model properly captures the V_i-to-ρ_i mapping it can be used accurately for any sequence length regardless of what length it was trained on.
To evalute the RNN network approach, we perform 50,000 training iterations on the $i = 0$ to $i = 100$ window of the training dataset, again with a batch size of 50 and Loss function $\mathcal{L}^{\rho_i}_{MSE}$. Again we also use the Adam optimizer, now with initial learning rate $\tau = 2 \cdot 10^{-4}$. The standard-RNN is trained with different dimensions of h_i.
The validation loss as a function of epochs can be seen in fig. 10a. The validation and training samples have similar MSE loss during training, so plotting the training loss provides no useful information. We find that the for $d \geq 3$ the loss of the models is greatly reduced. This indicates that the RNN model needs to pass on 3 variables to properly predict the density profile given the potential sequence. We find that the number of iterations, required before the cost/loss function converges, depends on the dimension d. Larger models need fewer iterations to converge. This is reasonable since a larger model means greater freedom in the choice of parameter combinations that can lead to a successful mapping.
The training process is also reliant on other factors like initialization of training parameters and training batch size. It takes 15000 training iterations on average before a reduction in the loss is observed. (See fig. 10a) However, a loss around 0.2 is still significant compared to the loss of models constructed later in the section. This tells us that the Standard-RNN has not captured the system as of 50000 iterations.
RNN models are notoriously difficult to train due to exploding/vanishing gradients. Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) RNN's introduced in [19] and [20] have additional parameters employed to tackle this problem. We are, however, looking for the simplest successful mapping to be able to extract meaningful information about the system itself. Hence, applying LSTM/GRU cells is not viewed as a viable option and will not be discussed further.

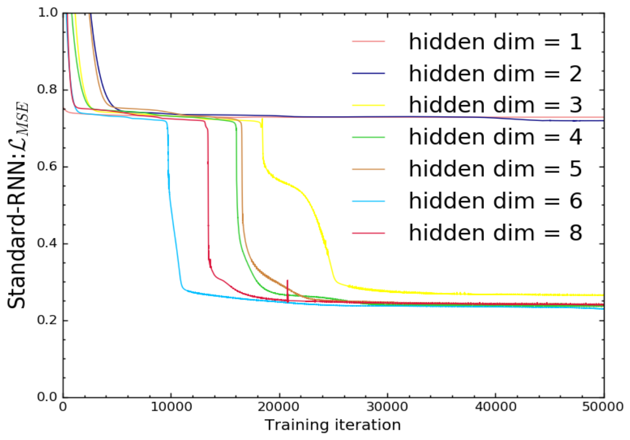
While the goal is not to find the most efficient training procedure, it is still required that the model can actually find the minimum of the Loss. Increasing the size of the model is an option, but will require an auxiliary architecture that assist in the extraction of physical features. The network is easier trained with d of the hidden unit being large, but that implies we are not teaching the simplest model imaginable, which increases the probability of the hidden state containing redundant information.
In order to find the simplest network that still yields an acceptable accuracy an alternative update law, motivated by [1], is employed.

Taylor update equation:

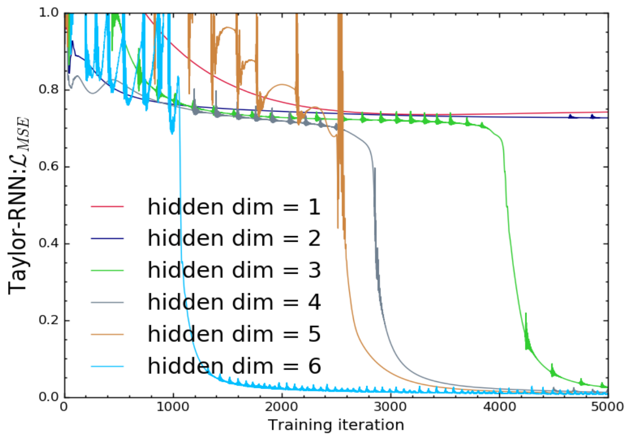
$$h_i = \sum_{n=0}^{n_f} W^{(n)} V_i^n * h_{i-1} \tag{36}$$

Output gate/ Projection:

$$\rho_i' = W^{(p)} * h_{i-1} \tag{37}$$



(a) Loss for 50000 training iterations for a standard cell RNN with different dimensions of the hidden state.



(b) Plot of Loss over 5000 training iterations for a RNN with a customized V_i-Taylor expanded update law. The loss is plotted for identical models only with different dimension of the hidden state.

Figure 10: Validation loss vs iterations for RNN networks with different update laws.

The new update law for h_i is a linear transformation of h_{i-1} with the $d \times d$ matrix $W(V_i) = \sum_n^{n_f} W^{(n)} V^n$. $W^{(n)}$ is a coefficient matrix for the n^{th} expansion in V_i. Even though, the transformation is linear in h_i, it is not linear in V_i since it is a sum of trainable weights multiplied with increasing powers of the input V_i. n_f is the chosen order of expansion. We refer to this as the $Taylor\ update\ law$. The output ρ_i' is obtained from the latent state h_i by a linear projection identical to the one used for the the standard RNN cell. The global structure of the RNN is identical to fig. 8, but the cell has changed. The Taylor-cell is shown in fig. 11. The W box represents the creation of the $W(V_i)$ matrix and $*$ is the symbol for matrix multiplication. The $Projection$ box represents the linear projection output gate.

The Taylor-RNN model is constructed with a 2. order expansion in V_i i.e. $n_f = 2$. The training of the new modelis done on the same sequence window as the standard cell ($i = 0$ to $i = 100$) and the training history is shown in fig. 10b for different sizes of the hidden state. The initialization of the training weights is particularly important for this recurrent cell. When the weights are initialized between 0 and 1 from a uniform distribution the loss often diverges on the first training step, making optimization impossible. The following initialization

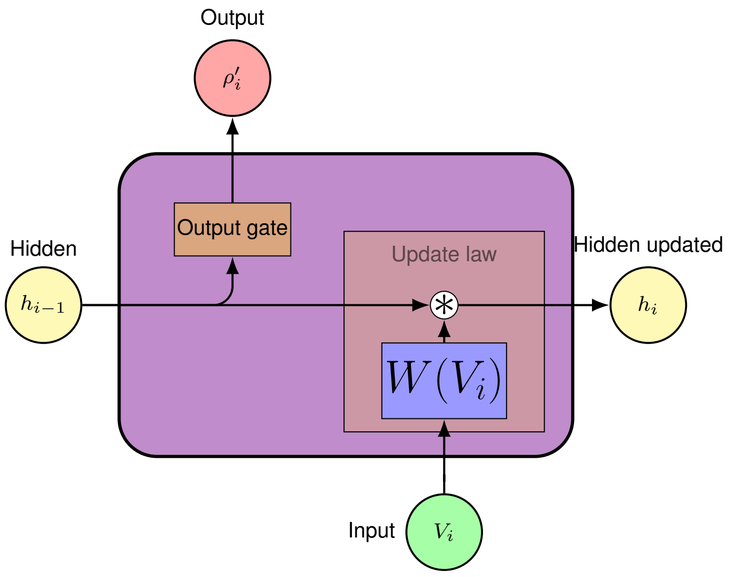


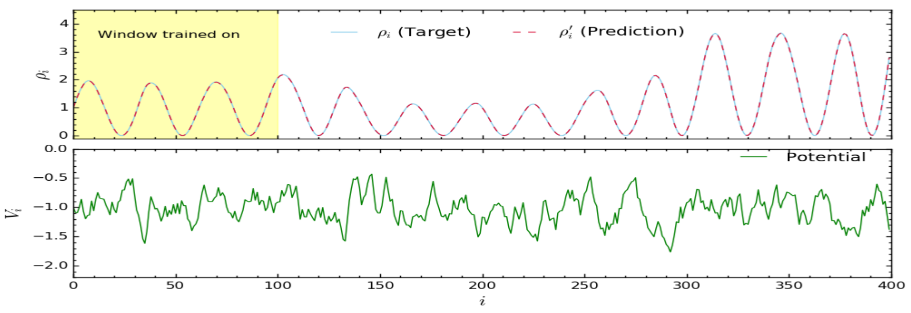
Figure 11: The structure of each recurrent cell with the Taylor expanded update law. Matrix multiplication is denoted by $*$.

avoids this issue:

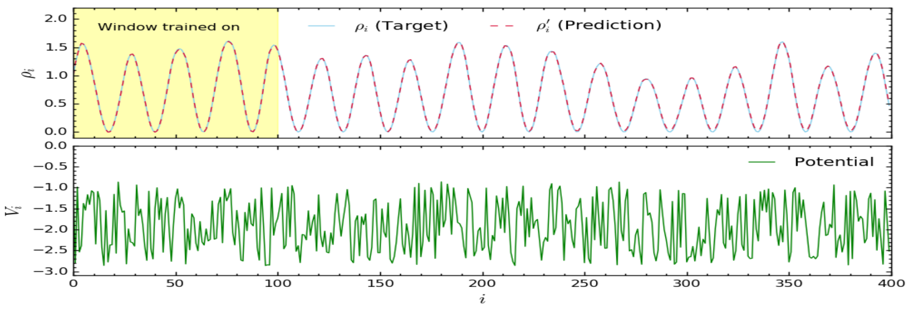
$$\begin{array}{ll} W^{(0)} = \mathbb{I} + \frac{0.01}{d} \times \text{randn}(d,d) \quad , & (\text{for } \ n = 0) \\ W^{(n)} = \frac{0.01}{d} \times \text{randn}(d,d) \quad , & (\text{ for } n > 0) \end{array}$$

Similarly to the standard RNN-cell (fig. 10a), loss is reduced for $d \geq 3$, further indicating that at least 3 features are needed for the model to perform a proper mapping. Comparing the loss vs iterations of the two different cell structures (fig. 10a and fig. 10b), the training steps needed for the RNN with the Taylor expanded cell is considerably lower than for the standard cell while reaching a superior performance. Thus, the Taylor expanded setup is clearly preferred. Additionally, the information of the V_i-expanded cell is easier to analyse since the $W^{(n)}$ matrices directly show the V_i-dependency for updating the hidden states of the network.
Testing for different values of n_f we find that for $n_f \geq 1$ the lowest sufficient values of d is unchanged. This indicates that the recurrent update equations have a linear dependency on V_i. This illustrates how finding the lowest n_f and d values without increasing the loss can provide essential information about the underlying physical system.

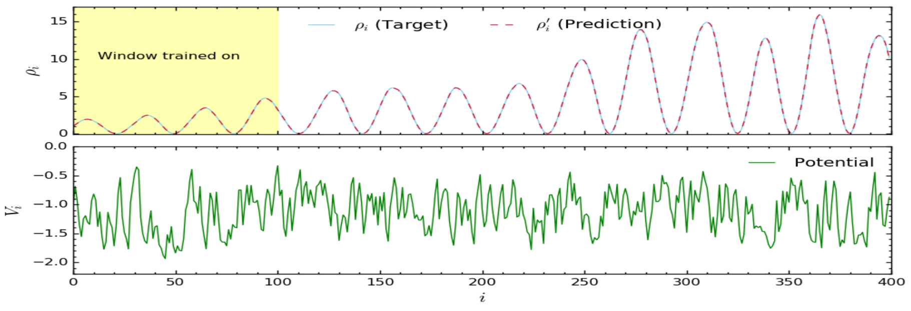
In fig. 12 the predicted density profiles and the simulated data for different test data samples is shown for a Taylor-model with $d = 5$, $n_f = 2$. Despite having dramatically different potential sequences, the Taylor-model is capable of predicting the density profiles with high accuracy over the entire $i = 0$ to $i = 400$ sequence. This is despite the model only being exposed/trained on a minor window ($i = 0$ to $i = 100$ of the training data. This showcase the raw predictive extrapolation power of the Taylor-RNN model. The relative error over the entire validation set is $\langle\Delta\rangle_{\text{Taylor}} = 0.2\%$ showing that the RNN significantly outperforms the much larger fully-connected models, where the 3-layer model had $\langle\Delta\rangle_3 = 11\%$.
Zhai et. al. [1] claims the relative error of their predictions to be less that 10%. Thus, the Taylor-RNN reproduces their result with significantly less error. However, it is difficult to compare the two results since we do not know exactly how they calculate the relative error.
The number of parameters for the RNN model with $d = 5$ and $n_f = 2$ is $(n_f + 1) \cdot d^2 + d = 80$. A very stark contrast to the huge and opaque fully-connected networks. The predictive power of the Taylor-RNN tells us that the machine must have obtained some knowledge about the underlying physics. The question is whether unnecessary information is still present.
To learn more we consider a Taylor-RNN for the lowest possible $d = 3$. The recurrent update law $h_i = \sum_n^{n_f} W^{(n)} V^n * h_{i-1}$ can be regarded as the networks formulation of the physical rules of the system. However,



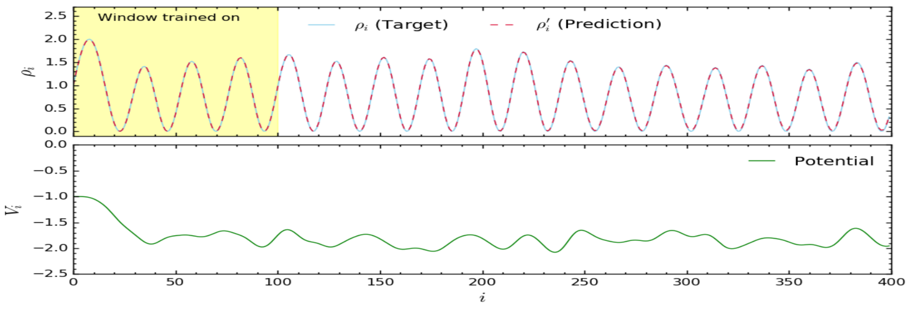
(a) Moderate variation in potential sequence.



(b) Rough and deep potential.



(c) Rough but shallow potential.



(d) Smooth potential sequence.

Figure 12: Examples of different output density sequences with their respective potentials compared to the simulated density profiles. The yellow region indicates the training samples window. All results are from test samples, which have not been trained on.

due to the output gate being a linear projection, we cannot be certain that this actually is the minimal number of variables needed to locally predict ρ_i. In this case, it turns out essential information about the governing physical equation would in fact be lost. Utilizing the information, we have about the simulated data, i.e. the Schrödinger equation (SE), we can compare it to the discoveries made by the RNN. The SE can be split into two 1. order equations, with

$$\partial_x \psi(x) = \partial_x \psi(x) \tag{38}$$
$$\partial_x(\partial_x \psi(x)) = V(x)\psi(x) \tag{39}$$

Which can be written on matrix form.

$$\partial_x \begin{pmatrix} \psi(x) \\ \partial_x \psi(x) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ V(x) & 0 \end{pmatrix} \begin{pmatrix} \psi(x) \\ \partial_x \psi(x) \end{pmatrix} \tag{40}$$

By considering the spatial coordinate, x, as discrete points, x_i, with spacing a, we obtain the discrete version of Eq.(39), the difference equations:

$$\Delta\psi_i = \frac{\psi_{i+1} - \psi_i}{a} \underset{\lim_{a\to 0}}{=} \partial_x \psi_i \tag{41}$$
$$\frac{\Delta\psi_{i+1} - \Delta\psi_i}{a} = V_i \psi_i \tag{42}$$

These equations result in a set of recurrence equations on a form, which can be implemented in the RNN architecture:

$$\begin{pmatrix} \psi_{i+1} \\ \Delta\psi_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & a \\ aV_i & 1 \end{pmatrix} \begin{pmatrix} \psi_i \\ \Delta\psi_i \end{pmatrix} \tag{43}$$

Recurrence equations are only comparable to SE for sufficiently small a, then $\Delta\psi_i \simeq \partial_x \psi_i$. The consequences of this restriction will be discussed in section 3.5. The set of equations show that only 2 complex features, namely the quantum wavefunction ψ_i and the first-order derivative $\partial_x \psi_i$ need to be stored for the Taylor-RNN model to perform the mapping of the system. However, one has to keep in mind that the current setup of the RNN is such that a linear combination of ρ_i must be kept in h_i. As a consequence, the update equations of the network might be done in terms of ρ_i instead. This makes a monumental difference since $\psi(x) \in \mathbb{C}$ (we are still not using $\psi \in \mathbb{R}$ for the considered system) has two real degrees of freedom. Whereas, $\rho(x) = |\psi(x)|^2 \in \mathbb{R}$ has only one real degree of freedom.
we have

$$\psi^*(x)\partial_x^2\psi(x) = V(x)\rho(x) \quad \text{and} \quad (\psi^*(x)\partial_x^2\psi(x))^* = \psi(x)\partial_x^2\psi^*(x) = V(x)\rho(x)$$

and hence, by the product rule

$$\partial_x^2\rho(x) = \psi^*(x)\partial_x^2\psi(x) + \psi(x)\partial_x^2\psi^*(x) + 2\partial_x\psi^*(x)\partial_x\psi(x) = 2V(x)\rho(x) + 2\partial_x\psi^*(x)\partial_x\psi(x) \tag{44}$$

It is impossible to lower the degrees of freedom of the system from two to one without losing information in the process. Hence, an additional variable is needed to express SE in terms of ρ_i.
The wavefunction is real for the current system. This means that two real numbers (ψ_i and ,$\partial_x\psi_i$) should be sufficient to solve the SE. As a result, we would expect that a $d = 2$ Taylor-RNN model could solve the system. Hence, $d = 3$ is not the most concise formulation of the system. But according to Eq. (44) even if $\psi(x) = \psi(x)^*$ the equation cannot be expressed purely in terms of $\rho(x)$ explaining why the Taylor-RNN alone cannot have accurate predictions with only $d = 2$.

Two explanations exist as for why the mapping is possible with $d = 3$. First, the model is solving the regular SE and storing ρ_i in the last hidden node. Second, it is possible to express the ρ formulation of the SE using 3 real variables (and using $\psi = \psi^*$).

The validity of the explanations can be examined by comparing the values of the hidden layer h_i for $d = 3$ and $n_f = 1$ to the simulated data for the specific case of all V_i = constant. Then, with $A_0 = B_0 = 1$, it follows from Eq.(27) and (28) that $\{A_i\} = \{B_i\} = 1$. This means that $\psi(x)$, $\rho(x)$ and $\partial_x\psi(x)$ take the analytical expressions:

$$\psi(x) = \cos(x) + \sin(x) \tag{45}$$
$$\partial_x\psi(x) = \cos(x) - \sin(x), \qquad \text{for } V = \text{const} \tag{46}$$
$$\rho(x) = \psi^2 = 1 + 2\sin(x)\cos(x) = 1 + \sin(2x) \tag{47}$$

$\psi(x)$ and $\partial_x\psi(x)$ both have a period of $T_\psi = 2\pi$ and their phase is shifted by $\phi = \pi/2$ i.e. $\psi(x + \pi/2) = \partial_x\psi(x)$. $\rho(x)$ has half the period of ψ. If ψ_i and $\partial_x\psi_i$ were present in h_i, we should see similar patterns. It is, however, challenging to directly verify this, since the RNN model's formulation of the update law is not restricted to a specific choice of basis.
There exist a set of equivalent recurrent equations all connected by linear basis transformations (General linear group, GL). In order to compare h_i values to the simulated results, we have to manually find a basis transformation that bring SE and h_i to their usual form. (Note the freedom in choosing a basis is rather important, since it makes the choice of initial hidden state h_0 irrelevant).
The principle behind the basis and similarity transformation can be found in appendix A.3. In fig. 13 the vector h_i with a specific linear transformation is plotted for constant potential. One can observe that the 3 hidden nodes (with or without basis transformation) all have the same period, π. Hence, ψ_i and $\partial_x\psi_i$ cannot be stored in h_i. If the hidden nodes contained linear combinations of ρ_i, ψ_i and $\partial_x\psi_i$ then h_i should only be periodic with a period of 2π. A superposition of periodic functions is only periodic if their periods have a common multiple. The least common multiple is the period of the superposition, which for a combination of ρ_i,ψ_i and $\partial_x\psi_i$ would result in a period of 2π. Therefore, this particular Taylor-RNN model cannot have discovered the quantum wavefunction ψ.

As proposed in Ref. [1], it is possible to formulate the SE in terms of ρ_i and two other real profiles $\eta_i = \mathrm{Re}(\psi_i^*\partial_x\psi_i)$ and $\xi_i = |\partial_x\psi_i|^2$ (assuming $\psi_i \in \mathbb{R}$). Which gives the following set of first-order differential equations:

$$\partial_x \begin{bmatrix} \rho(x) \\ \eta(x) \\ \xi(x) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 0 \\ V(x) & 0 & 1 \\ 0 & 2V(x) & 0 \end{bmatrix} \begin{bmatrix} \rho(x) \\ \eta(x) \\ \xi(x) \end{bmatrix} \tag{48}$$

The derivation of this follows from Eq.(44) and the derivatives of η_i and ξ_i. The derivation can be found in appendix A4. In the case of $\{V_i\}$ = const. $\eta(x)$ and $\xi(x)$ are given by:

$$\eta(x) = (\cos(x) + \sin(x))(\cos(x) - \sin(x)) = \cos(2x) \tag{49}$$
$$\xi(x) = (\cos(x) - \sin(x))^2 = 1 - \sin(2x), \qquad \text{for } V = \text{const} \tag{50}$$

All with the periods $T_\rho = T_\eta = T_\xi = \pi$, which correspond to the content of the h_i in fig. 13. In fact, we see that the values of the three nodes agree with the analytical expressions.
In fig. 14, h_i for one V_i sequence is compared to the simulated values of ρ_i, η_i and ξ_i showing that the network has indeed stored $h_i = (\rho_i, \eta_i, \xi_i)^T$ in the hidden state.
Additionally, we can show that for a specific training run of the Taylor-RNN model a similarity transformation of $W^{(n)}$ with $M^{-1}W^{(n)}M$, $M \in \mathrm{GL}(3,\mathbb{R})$ exists, such that the $W^{(n)}$ can be brought on the following form:

$$M^{-1}W^{(0)}M = \begin{bmatrix} 1.002 & 0.196 & 0.013 \\ 0.004 & 0.998 & 0.101 \\ -0.012 & -0.008 & 1.003 \end{bmatrix} \approx \begin{bmatrix} 1 & 2a & 0 \\ 0 & 1 & a \\ 0 & 0 & 1 \end{bmatrix} \tag{51}$$

$$M^{-1}W^{(1)}M = \begin{bmatrix} 0.012 & 0.000 & -0.002 \\ 0.103 & 0.014 & 0.002 \\ -0.012 & 0.195 & 0.014 \end{bmatrix} \approx \begin{bmatrix} 0 & 0 & 0 \\ a & 0 & 0 \\ 0 & 2a & 0 \end{bmatrix} \tag{52}$$

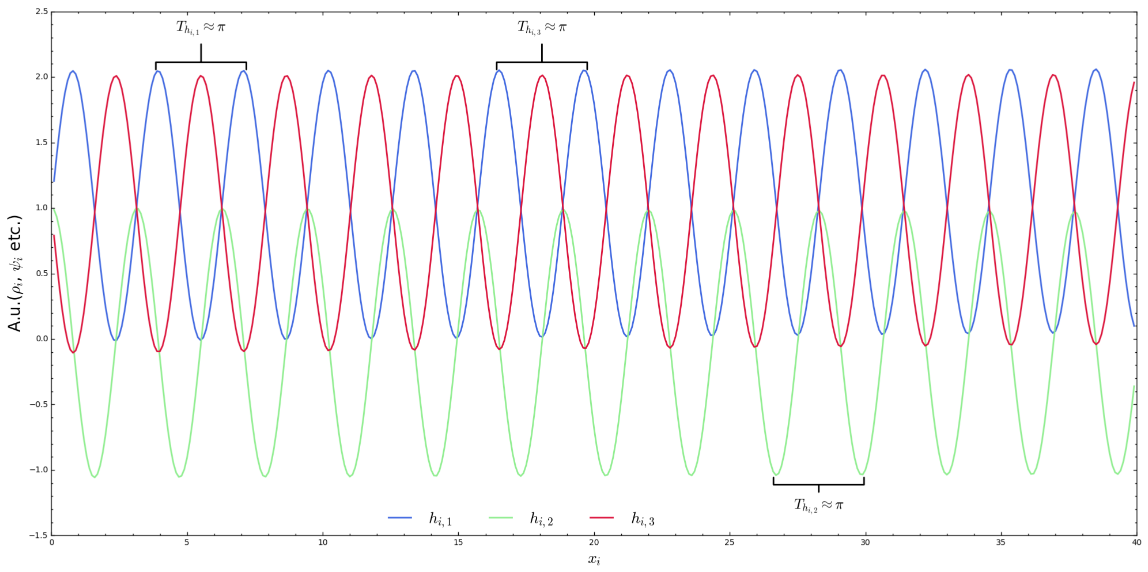


Figure 13: Trained Taylor-RNN (n_f = 1,$d = 3$) , plot of hidden states $h_i = (h_{i,1}, h_{i,2}, h_{i,3})^T$ vs. x_i for constant potential. All states have a period $T_{h_{i,n}} = \pi$ confirming that ψ_i and its first order derivative cannot be stored in h_i.

Which correspond to the difference equation

$$\begin{bmatrix} h_{i+1,1} \\ h_{i+1,2} \\ h_{i+1,2} \end{bmatrix} = \begin{bmatrix} 1 & 2a & 0 \\ aV_{i+1} & 1 & a \\ 0 & 2aV_{i+1} & 1 \end{bmatrix} \begin{bmatrix} h_{i,1} \\ h_{i,2} \\ h_{i,3} \end{bmatrix} \tag{53}$$

This is consistent with the discrete version of Eq.(48), if $h_i = (h_{i,1}, h_{i,2}, h_{i,3})^T$ is interpreted as $h_i = (\rho_i, \eta_i, \xi_i)^T$. Note that the the manual similarity transformation has nothing to do with the network training/learning. It is merely done so we can compare the results produced by the network to theory.
From this example, it is clear that the network can formulate rules of physics, such as Eq.(48), from the observed data. Due to the choice of projection output cell, the recurrent cell works directly with the variable of observation data to develop these laws. The downside of this is that higher-level concepts, like quantum wave functions, are not found this way. This will be further addressed in section 3.5 when the overall network architecture is discussed.

While meaningful information can certainly be extracted from the small n_f,d Taylor-RNN model, we are only able to find a description of the physical law in terms of output density. The network could not the discover the underlying ψ-description of the Schrödinger equation. To "teach" the machine about the quantum wavefunction a separate neural network architecture, called an Autoencoder (AE), will be employed. This new network is designed to extract and compress information from the RNN-model, which is solving the system. With this method, the RNN model does not have to focus on finding the lowest values of d and n_f. Instead, a network with larger hidden state size and V_i-expansion may be chosen, which is easier to train. The redundant information naturally present here, will be filtered by the AE. We consider this to be of even greater importance for a more complicated system, where identifying the simplest adequate solving model is close to impossible.

Conclusion

General RNN networks has been introduced, yielding a superior predictive power ($\langle\Delta\rangle_{\text{Taylor}} = 0.2\%$) compared to regular FCNNs ($\langle\Delta\rangle_3 = 11\%$) while at the same time using far less parameters. A Taylor expansion update

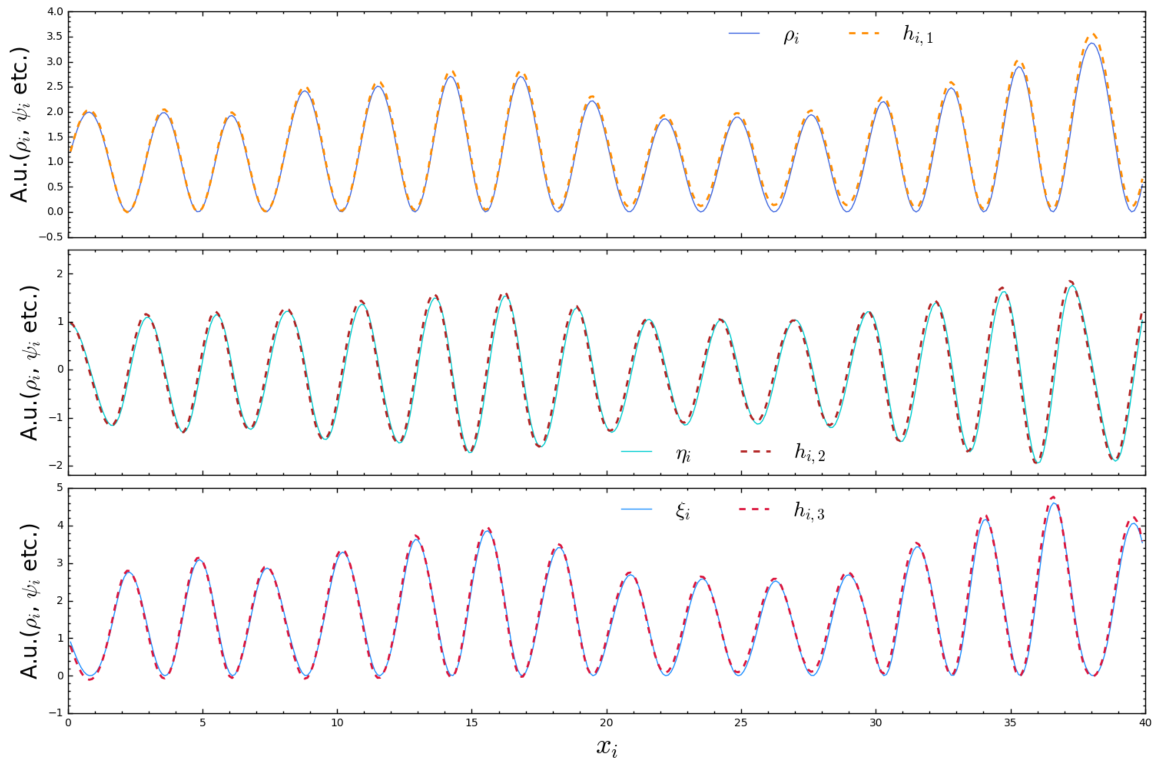


Figure 14: Plot demonstrating that hidden state of an arbitrary sample matches with ρ_i,η_i and ξ_i of the data.

law leads to easier training as well as providing key information about the system. However, an extended network architecture is needed to obtain deeper knowledge about the system.

3.4 Recurrent Autoencoder

The fact that RNN models are being capable of accurately predicting the density profiles, means that some information of the physics governing the system must be stored in the hidden states of the RNN. This essential information is, however, mixed with irrelevant information. To extract the "knowledge" from the h_i, we utilize a different neural network architecture, which is designed to learn from the neural activity of the RNN model. It works on the RNN hidden states to reduce the dimension of the hidden state and hence make the information of the network more relevant and compact. This network architecture makes use of so called autoencoders (AE), which are today commonly used tools for information compression or dimensional reduction, first introduced in [21].

3.4.1 Autoencoder introduction

Ref. [10] provides a good overview of autoencoders as a suplement to [21]. There exists various kinds of autoencoders, but they can be thought of as a special case of feedforward networks. Generally, AE's consists of two parts; an encoder function $h = E(x)$ and a decoder function $r = D(h)$,which attempts to reconstruct x. Nothing is challenging or particularly useful about having a feedforward network successfully learning to have $r = D(E(x)) = x$ everywhere. Hence, autoencoders have to be compromised in certain ways to make reconstructing x challenging. We will be using $undercomplete$ autoencoders, where the representation/state h is limited to always having a smaller dimension than x , written as $x \in \mathbb{R}^M$, $h \in \mathbb{R}^L$ with $M > L$. Learning to reconstruct x with an undercomplete AE drives the encoder to store only the most important features in h. The encoder and decoder functions are both realized by a feed-forward network.

3.4.2 Architecture setup

In order to obtain both the essential variables in h_i and the most efficient update rules of these compressed variables, the autoencoder function set is combined with a recurrent neural network structure and is then called a recurrent autoencoder (RAE). The RAE takes the hidden state, h_{i_0}, from the RNN-predictor as input at an arbitrarily chosen step i_0 and encodes it to the compressed hidden state $\tilde{h}_{i_0}$, where $\tilde{h}_{i_0} \in \mathbb{R}^{\tilde{d}}$. From there the RAE tries to reconstruct h_i for the following steps $i = i_0 + 1, i_0 + 2...$ by updating $\tilde{h}_i$ in a recurrent cell and then decoding the latent variables $\tilde{h}_i$. The architecture of the RAE is shown in fig. 15a. The update equations of the RAE structure is shown below.

Encoder:

$$\tilde{h}_{i_0} = E\left(h_{i_0}\right) \tag{54}$$

$$g_{i_0} = W^{(h_1)} * h_{i_0} + b^{(h_1)} \tag{55}$$

$$a_{i_0} = \mathrm{ELU}(g_{i_0}) \tag{56}$$

$$\tilde{h}_{i_0} = W^{(o)} * a_{i_0} + b^{(o)} \tag{57}$$

Recurrent cell:

$$\tilde{h}_i = \tilde{W}\left(V_i\right) * \tilde{h}_{i-1} = \sum_{n=0}^{n_f} \tilde{W}^{(n)} V_i^n * \tilde{h}_{i-1}, \quad (i = i_0 + 1, i_0 + 2, \cdots) \tag{58}$$

Decoder:

$$h_i' = D\left(\tilde{h}_i\right), \quad (i = i_0, i_0 + 1, i_0 + 2, \cdots) \tag{59}$$

$$\tilde{g}_i = \widetilde{W}^{(h_1)} * \tilde{h}_i + \tilde{b}^{(h_1)} \tag{60}$$

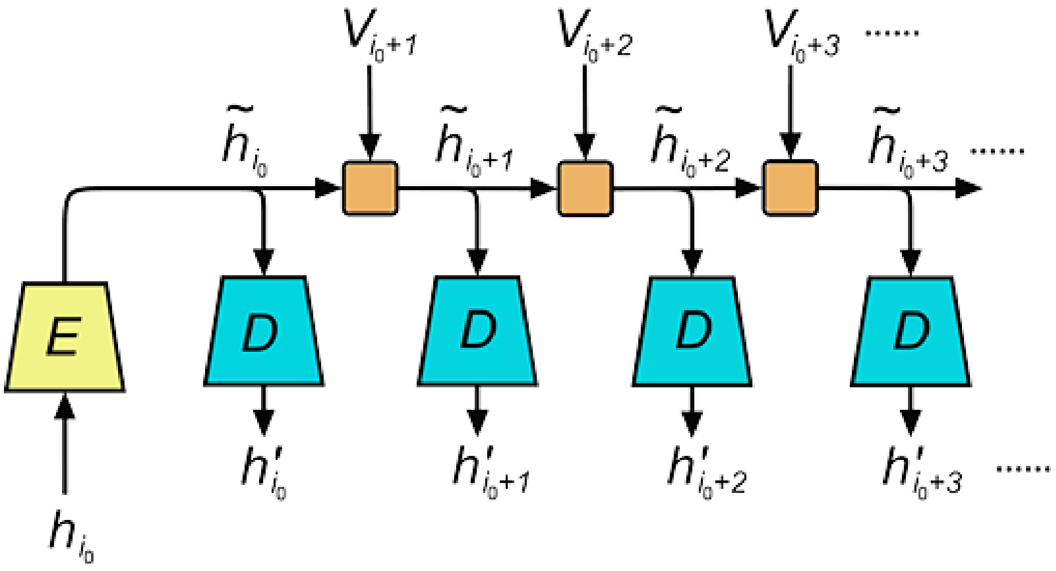
$$\tilde{a}_i = \mathrm{ELU}(\tilde{g}_i) \tag{61}$$

$$h_i' = \widetilde{W}^{(o)} * \tilde{a}_i + \tilde{b}^{(o)} \tag{62}$$

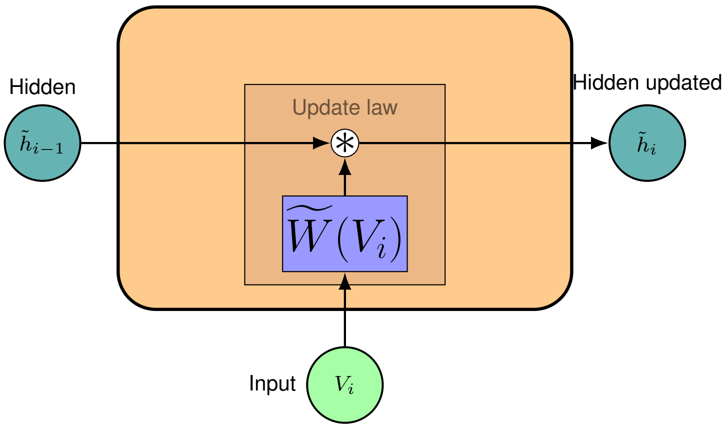
i_0 is an arbitrarily chosen number in the sequence and h_i' is the output of the RAE.
Utilizing an undercomplete AE means that $\tilde{d}$ must always be smaller than d. h_{i_0} is compressed with $E(\tilde{h}_{i_0})$, $\tilde{h}_{i_0}$ is updated at each step i, by the transformation $\tilde{W}\left(V_i\right) * \tilde{h}_{i-1}$, which is the expansion update law also used by the predictor. This step is visualized in fig. 15b. Finally, h_i is reconstructed with the decoder $h_i' = D(\tilde{h}_i)$. The RAE is evaluated by its ability to reconstruct the hidden states of the RNN-predictor. The training process consists as usual of minimizing the mean square error:

$$\mathcal{L}_{MSE}^{h_i} = \frac{1}{N_{\text{total}}} \sum_{i \in \text{sequence+batch size}}^{N} \left(h_i - h_i'\right)^2 \tag{63}$$

This type of training is usually labelled as "unsupervised", since we are not training the model on a simulated/-experimental (input/output) dataset but instead trying to reconstruct the hidden state profile, which is essentially an unknown set of d-dimensional vectors provided by a trained RNN model. Whether this training qualifies as unsupervised really depends on definition. The parameters of the model are still optimized based on the machine's ability to reconstruct h_i. In that regard all neural learning is supervised.
The RAE model is trained on hidden states from a trained RNN-predictor with $n_f = 2$, $d = 6$. In fig. 16 the training history is shown for different sizes, $\tilde{d}$, of $\tilde{h}_i$ but always with $d > \tilde{d}$. The reconstruction loss of the RAE increases dramatically when $\tilde{d} < 2$. This indicates that two variables is enough to store the essential features of the system.
Additionally, we claim that the two variables are actually the wavefunction ψ_i and the derivative $\partial_x \psi_i$. This claim can be verified using similar analysis methods as we did for the RNN-predictor models. First, we use the



(a) The RAE encodes/compresses the hidden state h_{i_0} to a lower dimension $\tilde{d}$. The compressed hidden states $\tilde{h}_i$ $(i = i_0 + 1, i_0 + 2, ..)$ is repeatedly updated in the RAE cell. After each update the decoder$D$ attempts to reconstruct the h_i corresponding to the step in the sequence.



(b) The RAE cell: Each step in the sequence, the compressed hidden state $\tilde{h}_{i-1}$ is updated with a linear transformation. The matrix $\tilde{W}(V_i)$, updating the hidden state is a sum of weight matrices multiplied by increasing powers of V_i.

Figure 15

RNN-predictor and RAE to find the density profile ρ_i and compressed hidden states $\tilde{h}_i$ for a constant potential $V_i = -1$. For V constant, the expressions for $\psi(x)$, $\partial_x\psi(x)$ and $\rho(x)$ are given by Eq. (47). The values can be compared to $\tilde{h}_i$ variables. The ρ'_i and $\tilde{h}_i$ profiles for $\tilde{d} = 2$ are illustrated in fig. 17. One can observe that the encoded hidden states have twice the period of the probability density. The hidden states have the period $T_{\tilde{h}_{i,1}} = T_{\tilde{h}_{i,2}} = 2\pi = T_\psi = 2T_\rho$ and have their phases are shifted by $\phi = \pi/2$. This matches the analytical expressions of ψ_i and $\partial_x\psi_i$ signifying that values of the latent variables $\tilde{h}_i = (\tilde{h}_{i,1}, \tilde{h}_{i,2})^T$ correspond to the state $(\psi_i, \partial_x\psi_i)$.
As previously mentioned in the analysis of the RNN models, the RAE also has freedom in the choice of basis used. Thus, the values of $\tilde{h}_i$ plotted have all been subjected to a common linear basis transformation M such that they are expressed in the standard basis.
The content of $\tilde{h}_i$ can also be verified for arbitrary potential sequences. $\tilde{h}_i$ (+ basis transformation) is plotted in fig. 18 with simulated ψ_i and $\Delta\psi_i = \frac{\psi_{i+1}-\psi_i}{a}$ for a random potential sequence from the test set. We see complete

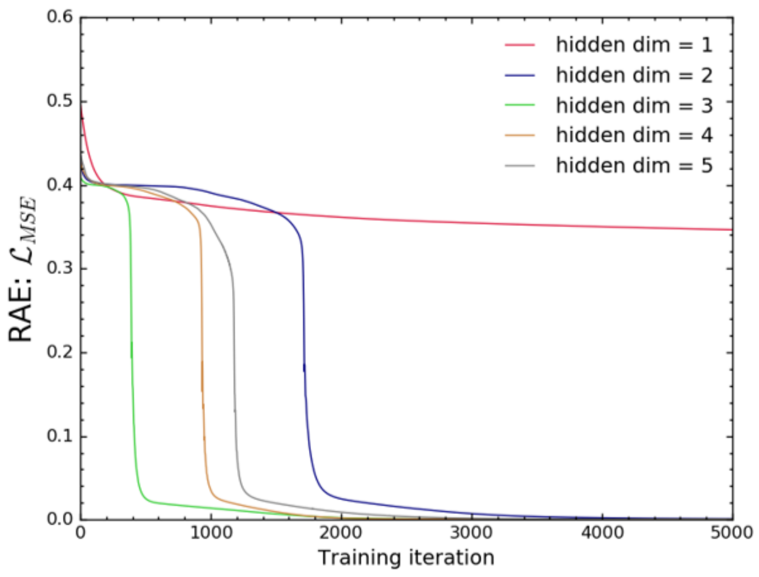


Figure 16: Loss vs. training steps for the RAE architecture for different sizes of h_i. Loss is significantly reduced for for $\tilde{d} \geq 2$.

correspondence between the content of the encoded hidden state and the simulated data further supporting that the RAE has stored ψ_i and $\partial_x \psi_i$ as the two features.
Finally, we examine the recurrent cell of the RAE used to update $\tilde{h}_i$. The RAE is already able to reproduce the hidden states of the RNN model given only the potential sequence and h_{i_0}. The weight coefficients $\widetilde{W}^{(n)}$ must be the networks formulation of the governing physical laws of the system. Because the update of the compressed hidden states is a linear transformation, all possible bases used to formulate the update law will be related by similarity transformations. We can manually find the change of basis matrix M that expresses the update equations in the standard basis. The weight coefficients yields:

$$M^{-1}\widetilde{W}^{(0)}M = \begin{pmatrix} 1.0019 & 0.1012 \\ -0.0010 & 0.9982 \end{pmatrix} \approx \begin{pmatrix} 1.0 & a \\ 0 & 1.0 \end{pmatrix} \tag{64}$$

$$M^{-1}\widetilde{W}^{(1)}M = \begin{pmatrix} 0.0029 & -0.0007 \\ 0.0965 & -0.0061 \end{pmatrix} \approx \begin{pmatrix} 0 & 0 \\ a & 0 \end{pmatrix} \tag{65}$$

The values of the rounded weights from Eq.(64) and Eq.(65) give us the following update equation:

$$\begin{pmatrix} \tilde{h}_{i+1,1} \\ \tilde{h}_{i+1,2} \end{pmatrix} = \sum_{n}^{n_f=1} \widetilde{W}^{(n)} V_{i+1}^{n} * \tilde{h}_i = \begin{pmatrix} 1 & a \\ aV_{i+1} & 1 \end{pmatrix} \begin{pmatrix} \tilde{h}_{i,1} \\ \tilde{h}_{i,2} \end{pmatrix} \tag{66}$$

The linear system of recurrent equation is consistent with the recurrent form of the discrete SE from Eq.(43) when setting $h_{i,1} = \psi_i$ and $h_{i,2} = \partial_x \psi_i$. Hence, we can infer that based on the data given the network architecture has identified the key features $h_i = (\psi_i, \partial_x \psi_i)^T$ and extracted the underlying physical law. For consistency, we find that the results can be recovered for different values of a, but only when $a << 1$. This matches with the results of Zhai et al. [1].

Conclusion

The RAE architecture employed to reduce the size of the network is in theory not needed since the RNN-model can be constructed in a way such that the same information is stored there. However, it is in practice a powerful

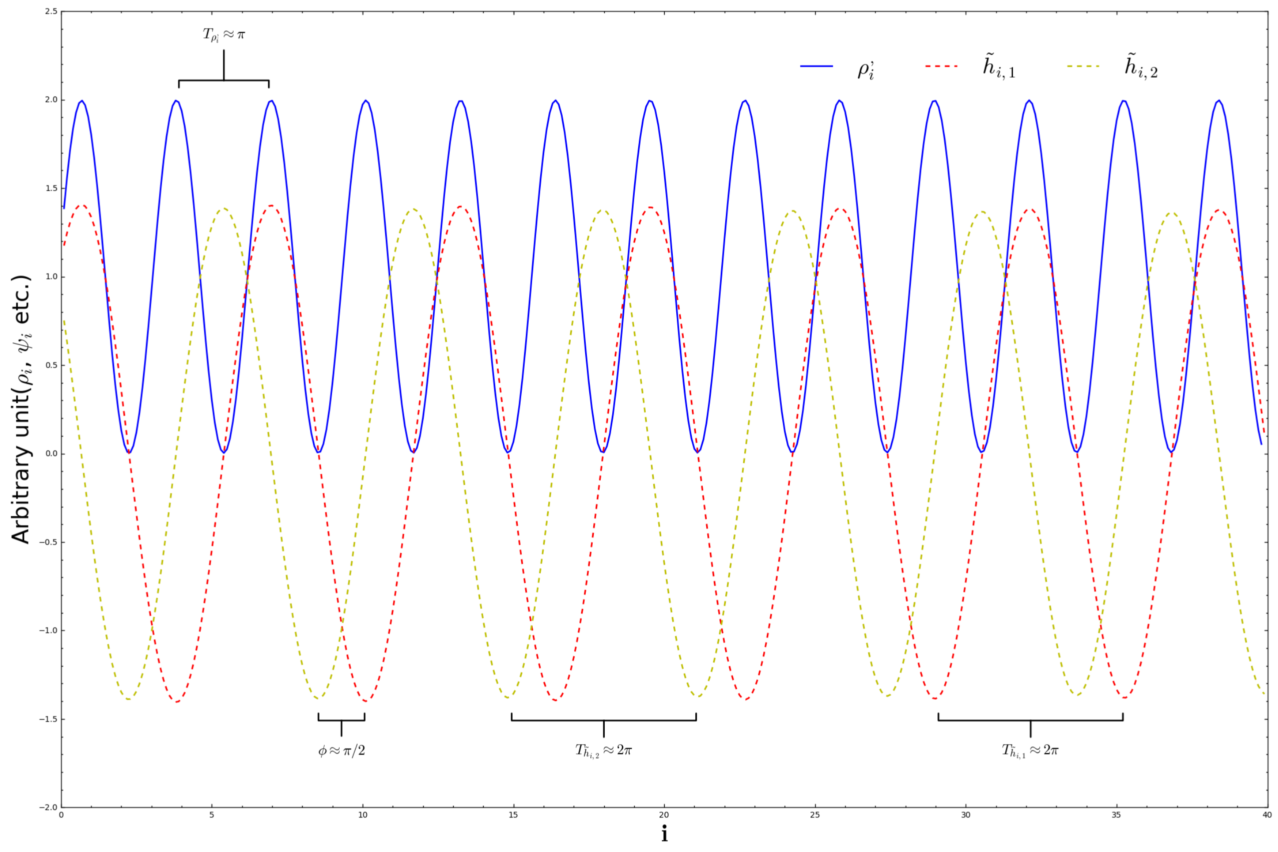


Figure 17: hidden states of the RAE and the output ρ_i' of the Taylor-RNN for constant $V_i = -1$. The values agrees with the values of the analytical expression of ψ and its derivative.

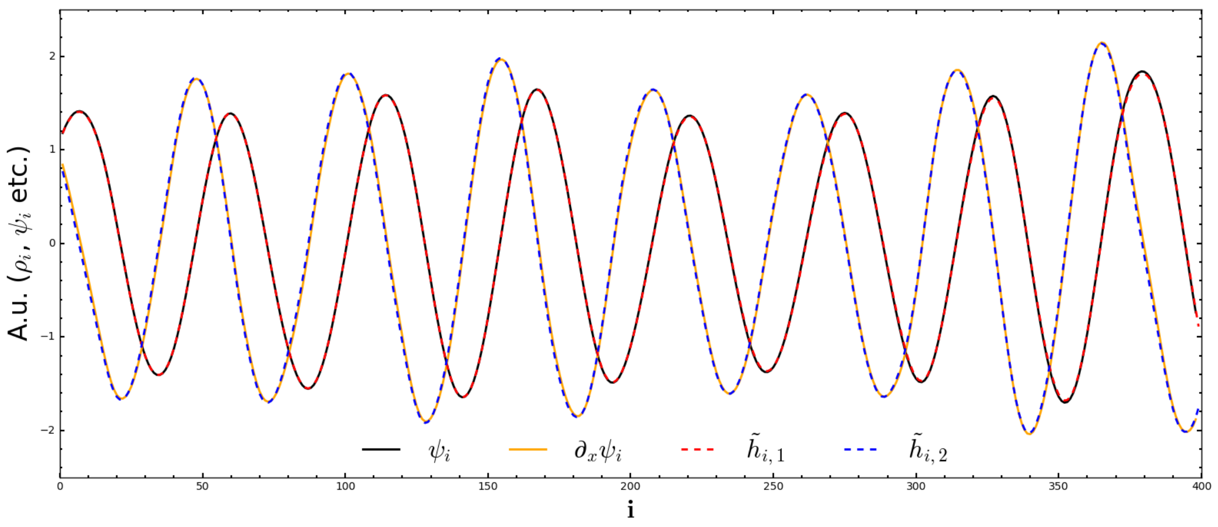


Figure 18: Comparison plot of hidden state values and simulated psi_i and $\partial_x \psi_i$ values for an arbitrary test sample.

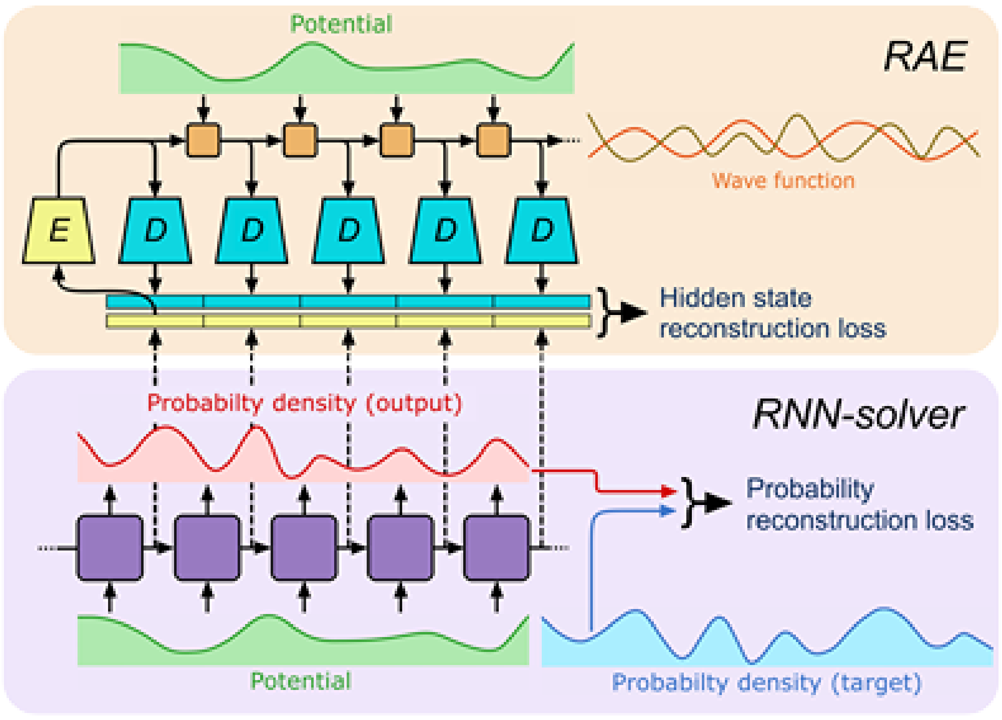

Potential
RAE
Wave function
E
D
Hidden state reconstruction loss
Probabilty density (output)
RNN-solver
Probability reconstruction loss
Potential
Probabilty density (target)

Figure 19: The architecture of the neural learning machine designed in [1] used to extract information about the laws governing a single quantum particle in 1D. The architecture consist of two major components. The RNN-solver (lower) is tasked with solving the potential-to-density mapping with minimal loss. The RAE (upper) is implemented to refine the information contained in the RNN-solver such that essential information of the system can be extracted.

tool, which can help discover underlying relations between different observables. We finish this section with a discussion on the strengths and limitations of the architecture.

3.5 Discussion

The neural architectures implemented in this section illustrates that machine learning is not strictly limited to regression and classification tasks. They can also be used to determine the most efficient representation of the underlying system, but is still a long way from unsupervised AI scientific discoveries with no human interaction. It should be mentioned, that when the output gate is changed from the projection gate to an FCNN, it is possible to train a $d = 2$, $n_f = 1$ Taylor-RNN. Indicating that the hidden states of the cell is able to store a combination of ψ_i and its first-order derivative, since the non-linear output gate can now successfully map $\mathrm{f}_{\text{out}} : h_i \to \rho_i$. In theory, this renders the RAE unnecessary, as the most parsimonious representation of the physical law can be found without implementation of the RAE. However, training of the $d = 2$ Taylor + FCNN output gate is very difficult. (See fig. 20. (left)) Even after 10^5 training steps the model has not yet fully converged . This is most probably due to vanishing gradient optimization issues. When the output gate becomes more complex, the gradients w.r.t. the parameters go to zero.
When different optimizations tools are used the $d = 2$ representation can be assisted/helped to find the minimal loss. For example by initializing the weight coefficients $W^{(0)}, W^{(1)}$ in the update equation with weight-values from the successfully trained $(d = 2)$ RAE. The assisted training history is shown in fig. 20 (right). Hence, gradually lowering the size of the model should allow us to extract the relevant information from just an RNN. However, we do find that the machine has an easier time learning the minimal representation with the implementation of the RAE.
Separating the architecture into one setup that performs the prediction and one that extracts the relevant information, has the upside that the information extraction process do not affect the performance of the network model.

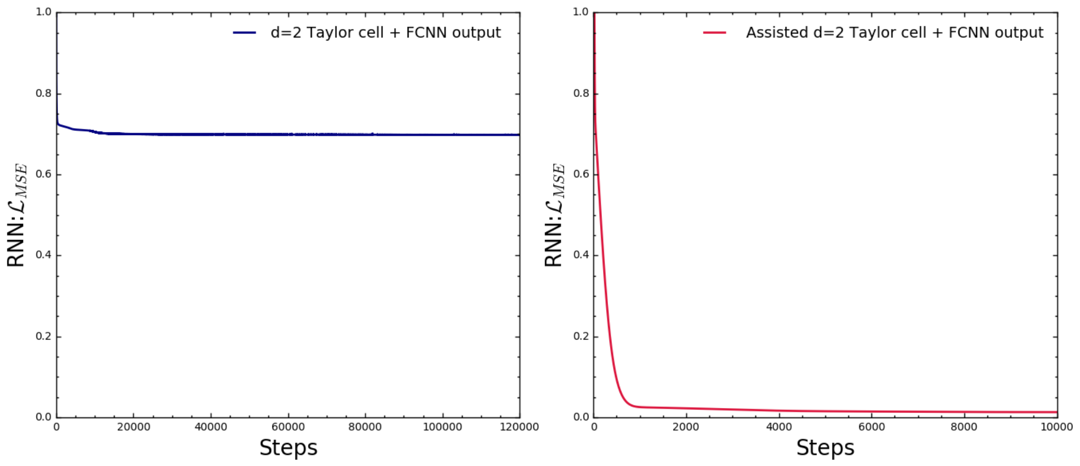


Figure 20: The two plots show that finding a Taylor-RNN representation with 2 variables is possible. With standard initialization of the cell parameters, the training fails (left). with specific assisted initialization the training converges (right).

We hypothesize that this could be related to the easier optimization process. The exact explanation for this is, however, not completely clear.
Regardsless, the approach of separating prediction and knowledge distillation provides for a more robust learning algorithm. This could be very valuable in other systems, where the most efficient representation of a task machine cannot be reached through training. Applying the network learning architectures to the simplified Schrödinger equation has taught us, that even though a parsimonious representation of a physical system exists, it is by no means easy to find. The difficulty increases as the neural models gets more complex. This competition between complexity and trainability remains one of the main issues gating neural networks from being of practical use in scientific discoveries aided by neural learning.
The developed network architecture based on a Taylor-RNN, shown in fig. 19 , is effective for the simplified Schrödinger equation problem, but only partially addresses more general issues. As we shall see further in the discussion, the use of Taylor expansion for the update law, becomes inefficient, if the complexity of the physical problem requires higher-order terms to be applied in the expression.

Other details about the architecture have to be discussed.
First, the initial hidden state provided to the RNN model. In the implementation of the model an all-ones vector is chosen as the first hidden state. With this, the RNN is able to predict the density, ρ_i, with very little relative error. However, when considering the physical variables stored in the hidden states, this should not be possible. The RNN is able to store a superposition of ρ_i (ψ) and its derivatives in the hidden states. Their values will naturally differ for different sequences of V_i. As a consequence, it is not sufficient to provide h_0 as the same constant for all samples.
The RNN-model works, in this case, because it is initialized with the same A_0, B_0 and V_0 for all samples in the dataset. This means that the first couple of values are almost the same in every sample, which makes a constant initial state sufficient. If A_0, B_0 and V_0 were changed between samples, we would have to provide a different h_0 depending on that. We will study an example of that in the next section.
Conveniently, this can be exploited to dictate which basis h_i and the update equation is formulated in. (However, only for the RNN!) If we set $h_0 = (\rho_0, \eta_0, \xi_0)$ (i.e. $d = 3$), the update equations of the Taylor-RNN is found on the form from Eq.(53) without the need for a basis change.

Second, the behaviour of the machine when subjected to different values of the potential cutoff a. Interestingly, we observe that the machine learns the discrete version of the SE (for $a = 0.1$). However, the physical system described in section 3.1 is an exact solution to the SE. The exact solution is generated from matching boundaries

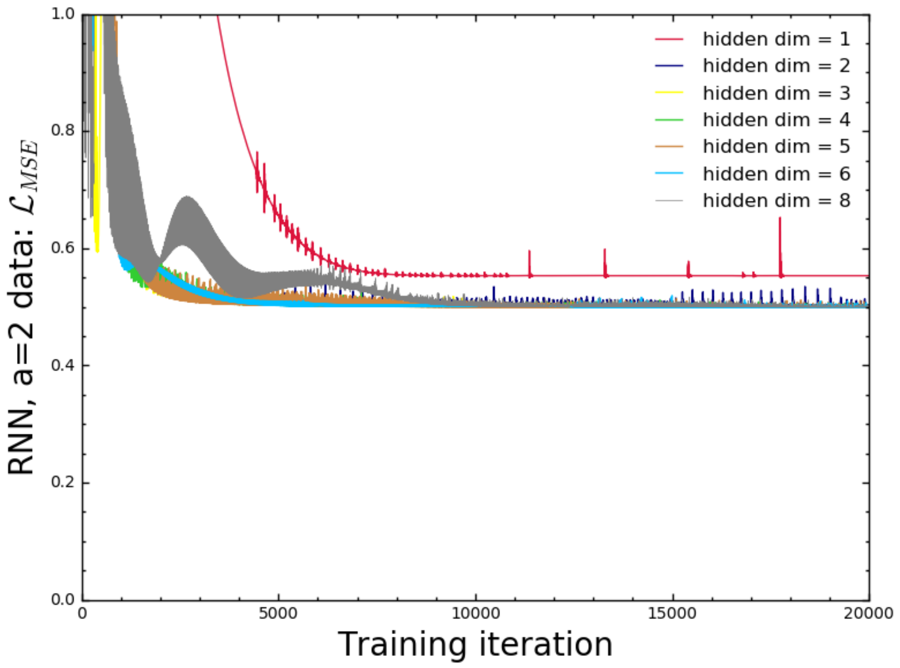


Figure 21: Loss during training for RNN models with the Taylor expanded recurrent cell and projection output gate for different hidden layer dimensions. The models cannot find a good potential-to-density mapping even for $d = 8$ and expanded to fourth-order in V.

at the cutoff a for each V_i. This means, that the machine actually finds a way to solve the problem, vastly different from the method used to generate the data. It turns out that the numerical solution (discrete SE) and the analytical solution, become equivalent for small a. The fact that the network finds a solution method that is different from the simulated data showcases the potential for machine learning aided scientific discoveries.

A dataset with $a = 2$ is generated to investigate what happens when the model is subjected to data where the discrete Schrödinger equation is no longer a good approximation. We find that the Taylor-RNN with projection output gate cannot be trained properly on this dataset. The training history[14] in fig. 21 shows how RNN models with hidden state dimension $d = 1, .., 8$ cannot reduce the MSE loss to an acceptable level, even when expanded to the fourth-order in V_i. We imagine this is because the essential variables change from ψ_i and $\partial_x \psi_i$ to A_i, B_i (and V_i). Apparently a n_f, $d = 8$ Taylor-RNN cannot approximate the recurrent equations from Eq.(27) and (28).
In order to approximate non-linear recurrent equations a 'universal' update law must be implemented[15]. An FCNN-based update law can potentially be used to determine the minimal number of variables necessary to accurately predict ρ_i. Consequently, we lose information about how the neural network solves the system. It is also more difficult to extract information from the hidden states, since the essential features stored here can be formulated in a completely arbitrary way. We study this challenge further in section 5.

The failure of Taylor-RNN to approximate non-linear update laws puts a ceiling on the potential applications for the learning architecture. A Taylor expansion can theoretically approximate any function arbitrarily well, however, practically there is a limit to the order that can be considered in the Taylor update law. The training of the model becomes increasingly difficult when higher order terms are included. The initialization of the weight parameters become very sensitive and can easily lead to divergent loss values. Additionally, the coefficient matrices which show the input dependency becomes harder to interpret when high-order terms are included. As a result, the transparency of the Taylor expanded cell is compromised. Based on this, the input expanded cell

[14]Training vs. epoch plot
[15]E.g. an FCNN with appropriate width and depth.

might only be usable for a limited set of systems.

Third, the neural network models could, without changing anything, be trained on a $(\{V_i\}, \{\psi_i\})$ dataset. The reason we have chosen to train on ρ_i instead of ψ_i is twofold. First, ρ_i corresponds to (if normalized) the probability of observing the quantum particle at a certain point in space. On the other hand ψ_i is not an observable quantity. Thus, thematically it makes sense using the ρ_i, since the (V_i,ρ_i) dataset just as well could be experimentally gathered data, we wanted to analyse. Second, if ψ_i was used all information about the system could be discovered by the task-solving machine alone. The RAE model would be unnecessary and we would consequently not demonstrate the true potential of the learning architecture.

As a conclusion on the studies in this section, we might have to re-evaluate the concept of an unassisted discovery of fundamental physical behaviour, including which kind of information we can hope to extract. On the one hand, the learning architecture has most certainly captured the underlying rules i.e. the discrete Schrödinger equation. On the other hand, the machine was in some ways "set up" for success. The only reason why, we could verify that the update laws of the network corresponded to the discrete Schrödinger equation, was because we selected the otherwise non-linear activation function to be linear in V_i. One has to imagine this was not intended when first applied in [1], but having any, non-proportional to V_i, weights in the RAE model, would make it impossible for the Schrödinger equation to emerge from the recurrent cell of the network.

4 Schrödinger Equation with Infinite Potential Boundaries

In the previous section, we saw the effectiveness of the RNN architecture when applied to a purely local system, where the probability density ρ_i could be determined from the potential V_i in that position. In this section, we will try to employ the architecture on a particular solution to the discrete Schrödinger equation (DSE) where infinite potential boundary conditions are enforced.
As a result, the wavefunction $\psi(x)$ is non-local and the entire potential (sequence) needs to be known in order to find the probability distribution and allowed energy of the quantum particle. The underlying law of physics (DSE) is, however, still local. We will test whether the RNN model developed can be expanded to accommodate for infinite wall boundary conditions, or if it will simply fail when applied to a non-local system.

4.1 Physical System

We start by introducing the physical system studied in this section. The concepts of the system are covered in standard quantum mechanics literature such as [18] and [22]. Once again, we consider the regular time-independent 1-D Schrödinger equation expressed in the continuous real space basis

$$-\frac{\hbar}{2m}\partial_x^2\psi(x) + V(x)\psi(x) = E\psi(x) \tag{67}$$

Except for certain simple given potentials $V(x)$, analytical solutions to this equation cannot be achieved. We will solve it numerically by transforming the differential equation into a matrix equation. The equation will then be an eigenvalue problem that can be solved through matrix diagonalization and afterwards the eigenstates ψ_n can be found through the characteristic matrix equation. In order to do this, Eq. (67) must first be cast on matrix form. This can be done, if a discrete basis is used instead of the continuous real-space position basis spanned by the states, $|x\rangle$.
To realize this, we begin by recalling some quantum mechanics formalism.
The Schrödinger equation in general operator form is:

$$H|\Psi\rangle = E|\Psi\rangle$$

where the quantum state $|\Psi\rangle$ can be expressed in the real space basis by using the complete set

$$\int \mathrm{d}x\, |x\rangle\langle x| = 1 \Rightarrow |\Psi\rangle \int \mathrm{d}x\; \psi(x)\,|x\rangle$$

with $\psi(x) = \langle x|\Psi\rangle$. For a complete discrete set of orthonormal basis states $|q\rangle$ (e.g. a complete discrete real-space basis), any state $|\Psi\rangle$ can then be expanded as

$$|\Psi\rangle = \sum_q C_q\,|q\rangle\,, C_q = \langle q|\Psi\rangle\,, \;\text{ with }\; 1 = \sum_q |C_q|^2$$

, which means the Schrödinger equation can be cast in any basis.

$$H|\Psi\rangle = E|\Psi\rangle = \sum_q C_q H\,|q\rangle = \sum_q C_q\,|q\rangle$$

$$\sum_k\sum_q C_q H_{k,q}\,|k\rangle = \sum_q C_q E\,|q\rangle\,, \quad H_{k,q} = \langle k|\,H\,|q\rangle$$

Using the orthogonality of the complete, set an equation for the coefficient of each state $|q\rangle$ is recovered (applying $\langle q|$ from the right).

$$\sum_q C_q H_{k,q} = E\sum_q C_q$$

For all q, the equations make up a matrix equation.

$$\begin{pmatrix} H_{11} & H_{12} & \cdots \\ H_{21} & H_{22} & \\ \vdots & & \ddots \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \vdots \end{pmatrix} = E \begin{pmatrix} C_1 \\ C_2 \\ \vdots \end{pmatrix} \tag{68}$$

Thus far everything is still exact. The solution becomes numerical, since we do not write the matrix in a complete discrete real-space basis. Instead, the continuous position x is turned into a grid of N discrete points each separated by $\Delta x = L/(N+1)$. Here L is the length between the boundaries. The finite basis results in an $N \times N$ Hamiltonian, which can easily be diagonalized. Fig. 22 illustrates the concept of the numerical solution. The exact wavefunction is shown in the blue graph. The crosses show the discrete points with a corresponding wave function value. As $\Delta x \to 0$ the numerical estimate of the wavefunction ψ_i becomes better and goes toward the exact solution. However, when the number of points N increases, obviously element count in the $N \times N$ matrix rise by power of 2 and diagonalizing the matrix becomes computationally demanding.
The expression of the Hamiltonian is affected when position $x \to \{x_i\}$ becomes discrete. We have already partially seen this in our analysis of the RNN models in section 3.3. The potential will simply be discretized into $V(x) \to V(x_1), .., V(x_i), .., V(x_N)$ i.e. a diagonal matrix. The derivative becomes a difference quotient like in Eq.(42) [16]. In the same way, the second-order differential operator can be approximated using a 3-point difference with Δ_x^2 given by:

$$\partial_x^2 \psi_i \approx \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{\Delta x^2} \tag{69}$$

Thus, the kinetic term $T = -\frac{\hbar^2}{2m}\partial_x^2$ becomes the tri-diagonal $N \times N$ matrix

$$T \to -\frac{\hbar}{2m\Delta x^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 \end{bmatrix} \tag{70}$$

We find the Schrodinger difference equation at each lattice point by combining the kinetic and potential term.

$$t_0 \left[2\psi_i - \psi_{i-1} - \psi_{i+1}\right] + \psi_i V_i = E\psi_i, \quad t_0 \equiv \frac{\hbar^2}{2m\Delta x^2}$$

Hence, for all N lattice points the equations make up the eigenvalue matrix equation $E\underline{\psi} = [H]\underline{\psi}$:

$$E \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_i \\ \vdots \\ \psi_N \end{bmatrix} = \begin{bmatrix} 2t_0 + V(x_1) & -t_0 & 0 & \cdots & \cdots & 0 \\ -t_0 & 2t_0 + V(x_2) & -t_0 & 0 & \cdots & \vdots \\ 0 & -t_0 & \ddots & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & 0 & 0 & \ddots & \ddots & -t_0 \\ 0 & \cdots & \cdots & 0 & -t_0 & 2t_0 + V(x_N) \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_i \\ \vdots \\ \psi_N \end{bmatrix} \tag{71}$$

All that remains is to modify the matrix $[H]$ such that the boundary conditions are accounted for. The infinite wall boundary conditions $V_0 = V_{N+1} = \infty$ means that $\psi_0 = \psi_{N+1} = 0$. This explains why the spacing is $\Delta x = L/(N+1)$, since it is actually the spacing between $N+2$ points. ψ_0 and ψ_{N+1} are, however, not included in the H-matrix. As a result, the ψ_1 and ψ_N will take the form

$$t_0 \left[2\psi_1 - \psi_2\right] + \psi_1 V_1 = E\psi_1 \quad , \quad t_0 \left[2\psi_N - \psi_{N-1}\right] + \psi_N V_N = E\psi_N$$

[16]The forward point difference

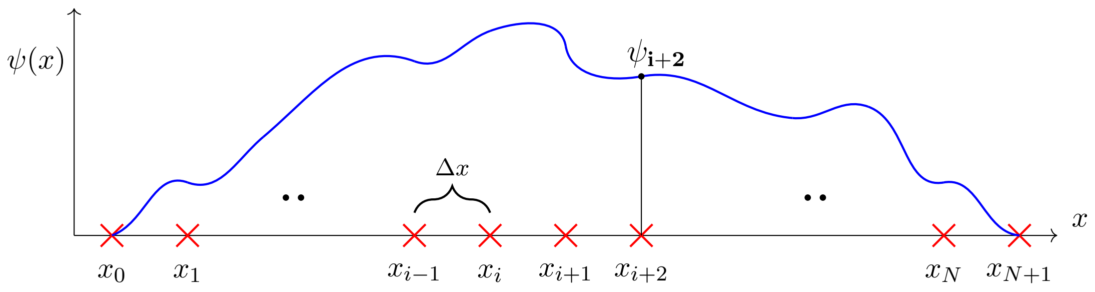


Figure 22: Sketch of discrete groundstate wavefunction. Keep in mind that this is not an entirely accurate plot of the wave function and is just meant to give intuition about the data generation. In reality the wavefunction is not smooth between that gridpoints (the wavefunction is merely the ψ_i dots). Δx is shown as the spacing between the discrete x_i position variables.

Incidentally, the matrix from Eq.(71) is naturally constructed such that these conditions are implemented. Other boundary conditions like periodic boundary conditions can also be done with some slight modifications to $[H]$. For convenience, the constant $\frac{2m}{\hbar^2}$ is absorbed into energy and potential, effectively setting $\frac{\hbar^2}{2m} = 1$ and $t_0 = 1/\Delta x^2$. Simplifying the Schrödinger equation (SE) Eq.(67) to

$$-\partial_x^2\psi(x) = (E - V(x))\psi(x)$$

This means that the amplitude of the potential is related to the mass of the quantum particle. This way, choosing the strength of the potential will be the only way to balance the relative energy cost of the kinetic and potential energy terms of the Hamiltonian.

4.1.1 Data Simulation

The discrete data is generated in the following way. N discrete potentials potential values $\{V_i\}$ are generated. Inspired by the previous system, a specific value of the potential is always repeated N_{repeat} times. Such that $N/N_{repeat} \in \mathbb{N}$ pillars of potential is generated. The height of each pillar is chosen randomly chosen between 0 and some upper bound V_{max}. Note for $N_{repeat} = 1$ it will just be 500 randomly generated potential $V_i \in [0; V_{\text{max}}]$ at position $x_i = \Delta x \cdot i$. Following this, the H-matrix is constructed. The eigenvalues E_n and eigenvectors $\{\psi_i\}_n$ are the found by diagonalizing the matrix and solving the characteristic equation. n is the excitation level of the solution. We will leave out the n-label for the rest of the section as we will mainly be dealing with the lowest energy $n = 1$ solutions. If excited states are used it will be clearly stated. Finally, the eigenstates are normalized such that: $\sum_{i=1}^{i=N} \psi_i^2 \Delta x = 1$. The probability density at x_i is the found as $\rho_i = \psi_i^2$. Since the bound states are normalized, ρ_i correspond to the probability of finding the particle at the position x_i.
In summary, the procedure of data generation is done in the following steps:

- Draw N/N_{repeat} discrete potential values V_i from a uniform distribution between 0 and V_{max}. The values are repeated for N_{repeat} points creating potential blocks each with length $N_{repeat}\Delta x$ which makes up the entire potential sequence $\{V_i\}$.
- The H-matrix is constructed as shown in Eq.(71)
- Eigenvalues $E_{(n)}$ and eigenstates $\psi_{i,(n)}$ are determined with a Lin Alg toolbox from NumPy library (see appendix B), that diagonalizes H-matrix. Density then found as $\psi_i^2 = \rho_i$
- Normalize the density/eigenstates such that $\sum_{i=1}^{i=N} \rho_i \Delta x = 1$ which is done by scaling $\psi_i/\sqrt{\Delta x}$.
- Select the eigenstates and corresponding energy for specified excitation levels. N eigenvalues/allowed energies are found for each H. The estimate of the exact solution gets increasingly worse for higher excitation.

Constants	N	N_{repeat}	L	V_{max}	Δx
	500	50	1	80	0.00199..

Table 2: Parameters/Constants used in the data generation of the Schrödinger equation with infinite wall boundary conditions.

Values of constants used in the data generation are shown in table 2. Testing has been done with different parameter values, but the ones listed are the values for data providing the core results of the section.
Note the choice of repeating V_i for $N_{repeat} = 50$ points and $V_{max} = 80$. This is because the $\{\psi_i\}$ produced should have a healthy variance. It means that the model trained on this data set will need to develop the proper relations to do the mapping and not find incomplete solutions. If the model is successful in discovering this mapping it should be able to perform the mapping for an arbitrary choice of $N_{repeat} \leq N$, even potentials constructed using a different method. The ratio N/L must be kept the same, since changing that would change Δx, which is part of the fundamental law (DSE) of the system. In consequence, the model would have to be retrained to adapt to such a change. Data for different energy levels can be constructed, but initially we will only consider the ground state.
We create 20000 different V_i and ρ_i sequences with $N = 500$ points in each sequence, 13000 used for training and 7000 used for validation. Like for the previous system, the validation data is withheld from the training process, and is subsequently used to analyse and evaluate the model. With the physical system setup and data generated, we proceed by subjecting different neural models to the data.

4.1.2 Batch Normalization

The V_i inputs have values in the range $V_i \in [0, 80]$. A wide range of input value is known to create unstable loss surface and hence, impedes the learning convergence. This is due to a phenomenon called internal covariate shift [23]. Which describes the fact, that changing the values of parameters in one layer affects the input distribution into the next one. We can combat this for the FCNNs by using the method known as batch normalization. In our case the input is scaled with the mean value of the entire training input data,

$$\mu_V = \frac{1}{N_{total}} \sum_{i \in \text{Training}} V_i \quad , \quad V_i \to V_i/\mu_V$$

, whereas the output target ρ_i not scaled. From a mathematical perspective, this appears highly questionable, since it corresponds to multiplying only one side of an equation (the mapping) with a constant i.e. creating an entirely new equation. However, in most cases, when applying neural networks, the goal is to find 'a' possible mapping, not a specific one. We can, therefore, utilize batch normalization for FCNNs, where the purpose is to find accurate predictions for the probability density. The actual $V \to \{\rho_i\}$ is concealed in the hidden layers anyway. This does not apply to our recurrent learning algorithm, where we look for the physical equations describing the system. In that case, we have to deal with a difficult training process.

4.2 Fully Connected Model

We apply FCNN models to find the probability distribution $\{\rho_i\}$ when provided with the potential sequence $\{V_i\}$. for this system is was sufficient to consider 1 hidden layer FCNNs. The models are constructed like the FCNNs in the previous section. We have trained models with different width i.e. number of hidden nodes ($d = 250, 500, 750$) to see, if d influences their predictive power. We continue using $\mathcal{L}^{\rho}_{MSE}$ as our optimization criterion and batch size is set to 100. Adam serves as the optimizer with initial learning rate $\tau = 1 \cdot 10^{-4}$. In fig. 23 we see, that significantly less training is needed for the models to converge compared to the previous system with unbound states. $\mathcal{L}^{\rho}_{MSE} < 10^{-4}$ can be reached for test data after $\sim$ 30000 epochs. This already indicates, that the bounded the system is more suited for an FCNN. The mean relative error $\langle \Delta \rangle$ from Eq.(30) is calculated

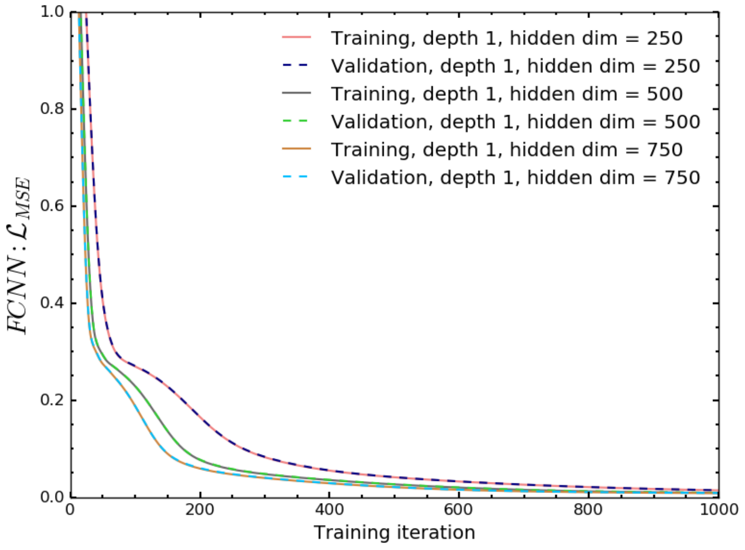


Figure 23: Validation and training loss for FCNN model, trained on ground state system, with 1 hidden layer and varying number of hidden nodes h_{dim}.

for the three different models.

$$\langle\Delta\rangle_{250} = 1.2\% \quad , \quad \langle\Delta\rangle_{500} = 1.0\% \quad , \quad \langle\Delta\rangle_{750} = 1.0\%$$

The three models have almost the same performance. Even the 'narrow' network with less hidden nodes than the input layer shows similar performance to the 500 and 750 width models. In fig. 24, the prediction of the 250 and 750 width model is plotted with the target density ρ_i for 2 randomly chosen potential sequences from the test set. Similar to the relative error, the plots show that both models can perform the potential-to-density mapping well and with only minor improvement when using the wider model.
Unlike in the previous section[17], the GS solution, for each potential sequence, has a particular allowed energy. Therefore, we can also train an FCNN to a $F : \mathbb{R}^N \to \mathbb{R}$ $\{V_i\}$-to-E mapping. Such an FCNN model is fairly painless to set up and train and will therefore, not be shown in the thesis.
Instead, we look to set up an FCNN, which can combine density and energy predictions. The mapping then becomes $F : \mathbb{R}^N \to \mathbb{R}^{N+1}$. This network is also straight forward to implement as the network take a similar form to fig. 3. where $y' = (\rho_1, .., \rho_i, .., \rho_N, E)^T$ and the loss function is chosen to be:

$$\mathcal{L}_{MSE} = \mathcal{L}^{E}_{MSE} + \mathcal{L}^{\rho_i}_{MSE}$$

The training for this model is shown in fig. 25 and prediction of probability distribution $\{\rho_i\}$ and E is shown for 2 test potentials in fig. 26. We see, that both E and ρ_i can be predicted with $\sim 1\%$ relative error and $\mathcal{L}^{E}_{MSE} < 10^{-5}$, $\mathcal{L}^{\rho_i}_{MSE} < 10^{-4}$, respectively. This performance is similar to having FCNNs of comparable size do the mapping ρ_i and E individually. Thus, a similar size model can predict predict both energy and density simultaneously, without sacrificing loss, indicating that a connection between the energy and density mapping exists.
In conclusion, the combined FC model can predict both energy and density to an acceptable level. , it is, however, done entirely by supervised learning. We acquire no underlying information about the system.
Accurate predictions like this could serve as alternatives to numerical methods or at least as a qualified first estimate to ground-state solutions of quantum problems. But, we are here more interested in whether the allowed ground state energy can emerge unsupervised from a learning architecture.

[17]There we chose particular solutions, where the energy was absorbed into the potential, setting energy to zero.

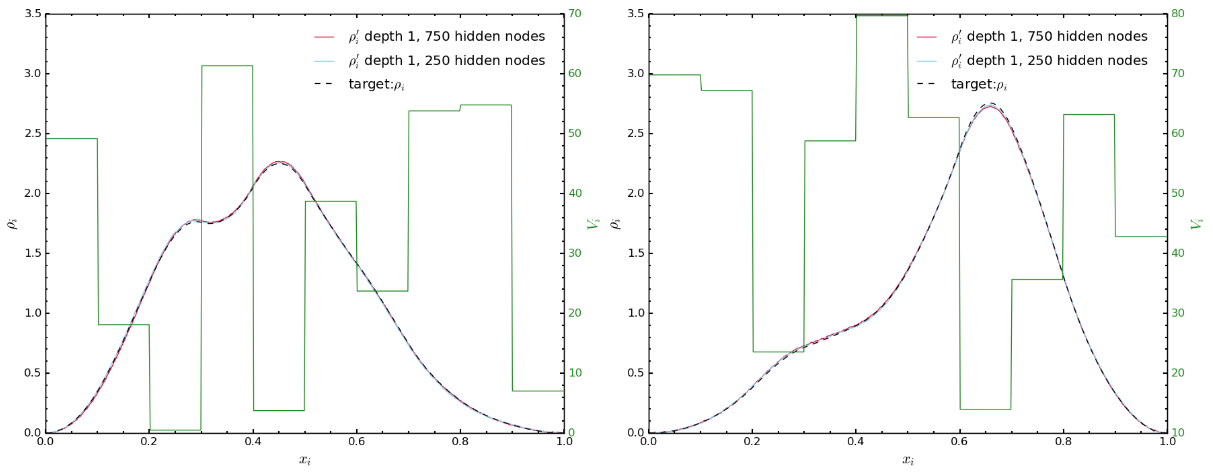


Figure 24: Plot of FCNN (ground state system trained) model prediction for 2 different test sample potentials. Only a minor improvement in prediction is observed for the the 750 width model compared to the model with 250 nodes.

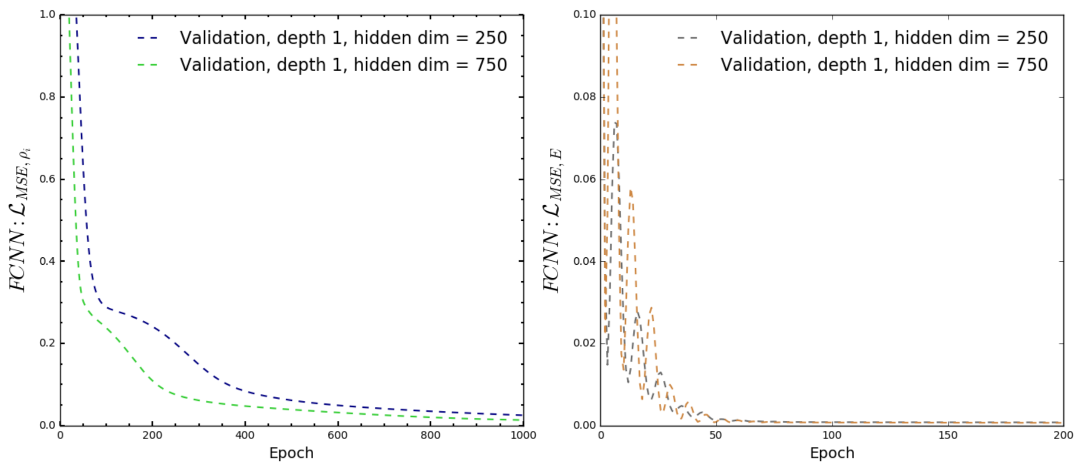


Figure 25: Loss for the combined FCNN model (ground state system trained). The validation loss is quickly reduced for both energy and density.

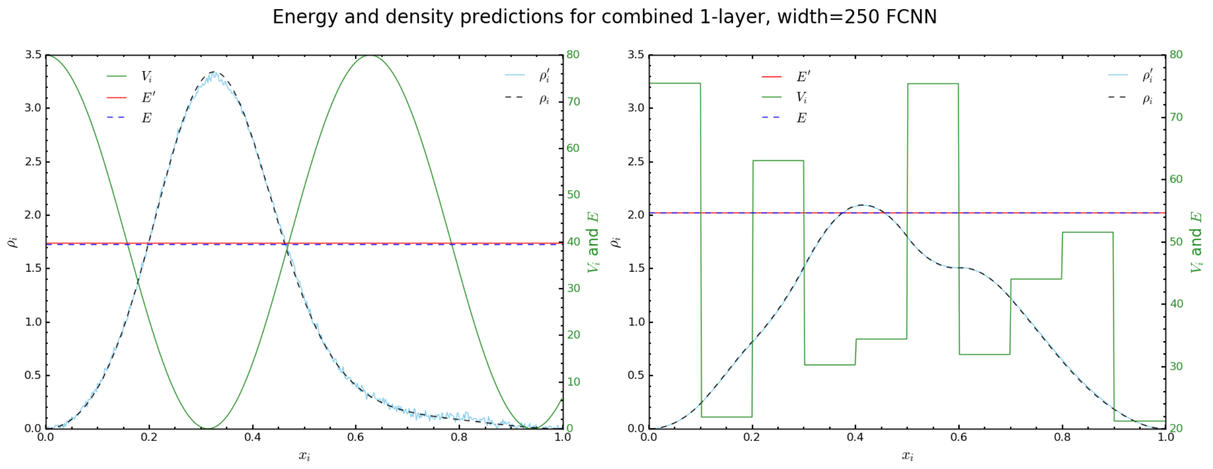


Figure 26: Comparison of target density and output from the combined FCNN model (ground state system trained) for 2 potential sequences. Left) Periodic potential used to show that the model can accurately predict E and ρ_i when subjected to potentials different from the test and target samples. Right) Arbitrary test set potential.

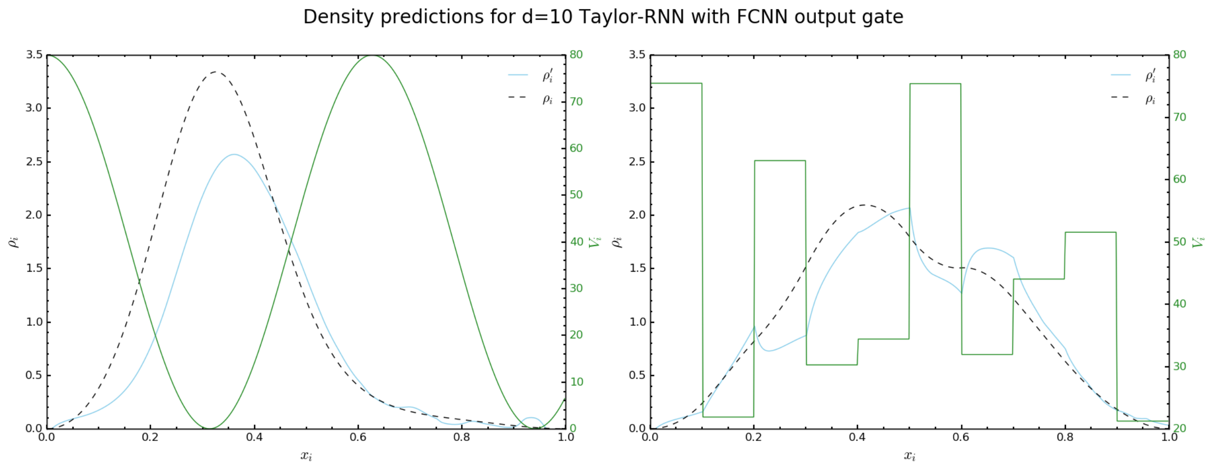


Figure 27: d=10 Taylor-RNN (ground state system trained) target and prediction comparison for 2 potential sequences. Left) shifted cosine potential. Right) Potential sequence from the test set.

4.3 Recurrent Neural Network Approach

Like for the previous system, we first try the somewhat naive approach to reduce the number of model parameters by utilizing an RNN model to perform a sequence-to-sequence mapping. We try both a conventional cell and a Taylor expanded cell with both an FCNN output gate and projection output gate, trained for 100000 epochs and default Adams optimizer setttings and batch size. It turns out, however, that none of them can reach a performance close to regular FCNNs. This is done using the same batch normalization as for FCNNs, which is technically not the correct approach due to reasons explained above, but this turns out to be of no consequence, since even the RNN models fail here.
In fig. 27 predictions are compared from a ($d = 10$, $n_f = 2$) Taylor + Projection cell to the target values. The same test potentials as in the combined model are used. We observe poor prediction compared to combined FCNN. The model has validation loss $\mathcal{L}^{\rho_i}_{MSE} = 0.10$ and $\langle \Delta \rangle \simeq 27\%$. Naturally, as the size of the hidden state is increased, the RNN models manage to find some inadequate estimate of the mapping, but is never close to the performance of the fully connected ones.

4.3.1 Extended Recurrent Architecture

The RNN models fail to accurately estimate ρ_i regardless of cell structure. FCNNs, in contrast, can predict the ρ_i with little loss. This tells us that ρ_i cannot be found from local input only. Information about the global structure (entire input sequence) is needed. To provide that, we propose an additional neural component to the learning architecture, called the Global Encoder (GE). The GE takes the entire potential sequence V as input. The depth and width of the global encoder are adjustable, but 1 hidden layer and 400 nodes are found to be sufficient for this system. The Global Encoder is responsible for two tasks. Deciding the initial hidden state h_0 of the RNN and encoding the global variables z of the system.

Global Encoder:

$$[z, h_0] = W^{(z)} * (\text{ELU}(W^{(h1)} * V + b^{(h1)}) + b^{(z)} \tag{72}$$

Where $z \in \mathbb{R}^{d_z}$ is a d_z dimensional vector and $h_0 \in \mathbb{R}^{d}$. Obviously $d_z < N$ must apply, but to extract the most meaningful information d_z should even be kept as small as possible without compromising the loss. The global variables are fed as an additional input to the recurrent network. Hence, the introduction of the GE means we have to modify the recurrent cell of the learning architecture.
We propose a new update equation inspired by the multivariable Taylor expansion (MTE). The update equation defined by a finite multivariable power series is given by

RNN update equation:

$$h_i = W(V_i, z) * h_{i-1}, \quad W(V_i, z) = \sum_{n_V=0} \sum_{m_1=0} \cdots \sum_{m_{d_z}=0} W^{(n_V, m_1, .., m_{d_z})} V_i^{n_V} z_1^{m_1} \cdots z_{d_z}^{m_{d_z}} \tag{73}$$

$W^{(n_V, m_1, .., m_{d_z})} \in \mathbb{R}^{d \times d}$ is the weight matrix for expansion to the n_V^{th}-order in V_i and $(m_j)^{th}$ order in the global variable z_j etc. We can write the update equation in a more compact way by applying multi-index notation, $n = (n_V, m_1, .., m_{d_z})$ and $\zeta = (V_i, z_1, .., z_{d_z})$. Such that $W^{(n_V, m_1, .., m_{d_z})} = W^{(n)}$ is the n^{th}-order coefficient matrix. The two useful properties of multi-index notation;

$$|n| = n_V + m_1 + \cdots + m_{d_z}$$

and

$$\zeta^n = V_i^{n_V} z_1^{m_1} \cdots z_{d_z}^{m_{d_z}}$$

result in a compact formulation of the n_f^{th}-multivariable Taylor expansion matrix

$$W(V_i, z) = \sum_{0 \leq |n| \leq n_f} W^{(n)} \zeta^n \tag{74}$$

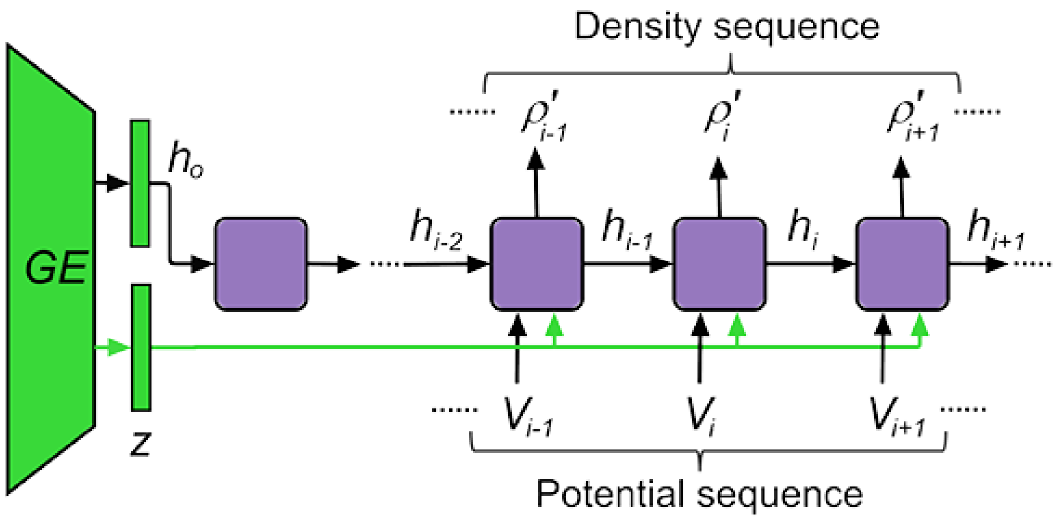
The output gate remains the same in the extended RNN cell. ρ_i' is generated from a linear projection of h_{i-1} as

Projection:

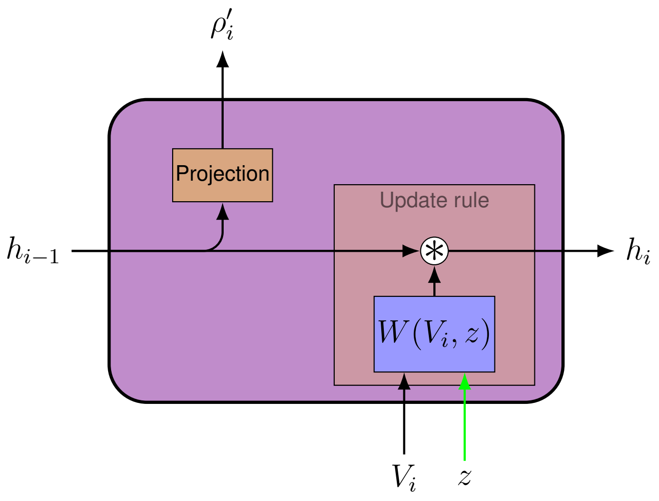
$$\rho_i' = W^{(p)} * h_{i-1} \tag{75}$$

The setup of a GE+RNN model is illustrated in fig. 28a and the recurrent cell shown in fig. 28b.
With this, we are ready to train the extended model on the dataset. We construct a model with $d = 5$, $n_f = 1$ and $d_z = 1$. However, the training process is more complicated this time. We cannot use batch normalization if we want to discover the underlying rules of the system. Hence, we have to train the model with unscaled data. The large V_i values lead to a divergence of the loss on the first iteration leaving us unable to continue training. We can deal with this by initializing the weights with values $<< 1$, but the training procedure is very unstable. The training process can be described as traveling along a very narrow mountain path with a canyon on both sides[18]. The slightest misstep (wrong adjustment of the parameters) results in the loss exploding and the optimizer giving up.
Fortunately, we have other tools at our disposal to make the training possible. One alternative, is the method of optimization by annealing, originally introduced in Ref. [24]. This can be adapted to neural learning where the learning rate starting from $\tau = 2 \cdot 10^{-5}$ is gradually reduced during training iterations. In addition to this we also set a criteria to stop the training early if the loss is reduced to $\mathcal{L}_{MSE}^{\rho_i} < 10^{-3}$. Even with these measures the training is very unreliable and the optimizer is likely to 'misstep' causing a spike in loss which potentially leads to a "killing" of the weights such that any prediction is returned as 0 (The MSE for this is only ~ 1). Therefore, if the loss stays exactly the same for 100+ iterations, the model is re-initialized and training starts over.
When a model finally reaches $\mathcal{L}_{MSE}^{\rho_i} < 10^{-3}$ we transition into a different training phase: the alternating training phase (fine-tuning). Here the RNN cell and the GE parameters are trained separately i.e. the RNN parameters are kept constant when the GE parameters are adjusted and vice versa. With these measures, we manage to successfully train the $d = 6$, $n_f = 1$, $d_z = 1$ model to reach $\mathcal{L}_{MSE}^{\rho_i} < 10^{-4}$.
The training history is displayed in fig. 29. One curve (red) shows a failed training run, but the training run illustrated by the blue curve successfully manages to find a potential-to-density mapping. Furthermore, we observe that the loss converges for $d_z = 1$. This tells us that only one global variable is needed to describe the model (We already discovered with the regular RNN that $d_z = 0$ does not work). Likewise we conclude, that the local mapping is linear in V_i and z, since the model can accurately predict the density distribution using a first-order MTE ($n_f = 1$).

[18]The canyons here representing failed training and not minimal cost areas.



(a) The combined model of the global encoder and RNN-solver. The global encoder takes in the entire potential to determine essential variables for the system that is the used additional input in the recurrent cell.



(b) The extended RNN-solver cell: The update equation is also an expanded in the global variable(s). The output is a linear projection of the previous hidden state h_{i-1}. The updated hidden state h_i is a matrix product of MTE matrix $W(V_i, z)$ and h_{i-1}.

Figure 28

The relative error of the $d_z = 1$,$d = 6$,$n_f = 1$ model found to be $\langle\Delta\rangle = 1.1\%$ and predictions for 4 different potential sequences are shown in fig. 30 . The accuracy of the model is comparable to the FCNNs and much better than the simple RNN model. The model could have been trained further for greater accuracy, but we find this level adequate to show that the underlying physics of the system has been captured.
In fig. 30c and fig. 30d the model is exposed to potentials created differently from the training and validation set. We compare this to fig. 26 where the FCNN model is subjected to the same potential. Although they roughly predict ρ_i equally well, we notice a distinct difference. The predictions made by the FCNN become noisy for potentials different from the training data. Whereas the output sequence for the extended RNN model is smooth regardless of the potential input given. Thus indicating that the RNN model has captured the underlying physical mapping in a more robust way than the fully connected network.
Additionally, the network model can discover the DSE from a set of potentials with the same overall construction method (but still from a diverse dataset). This is demonstrating the potency of RNN, which (due to the limited

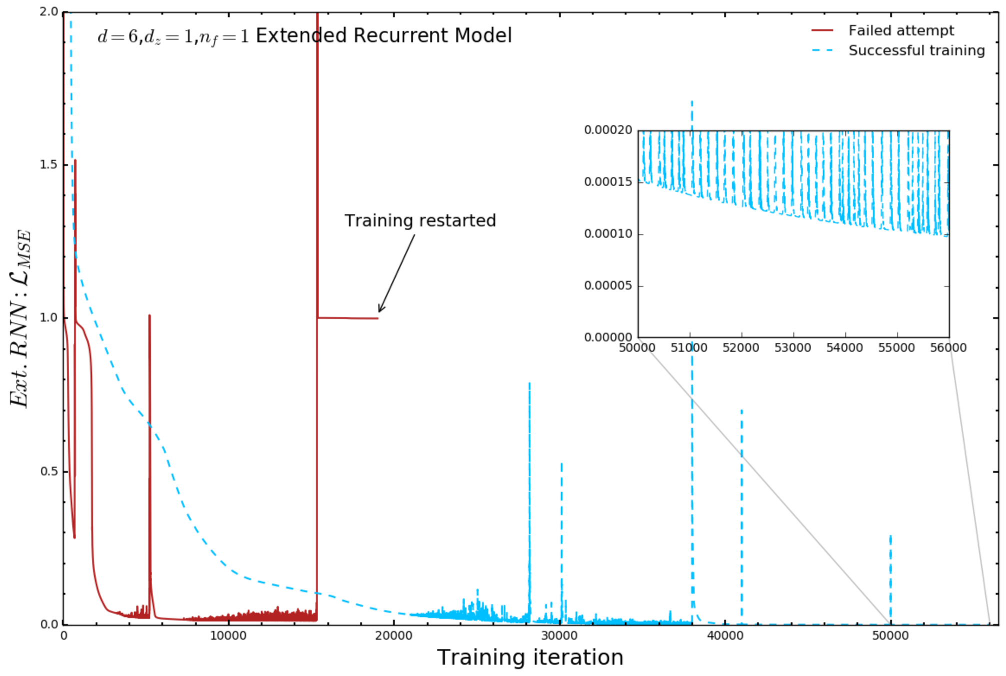


Figure 29: Two training runs of the extended RNN design with $d_z = 1$,$d = 6$,$n_f = 1$.Only difference is the random initialization. One training run fails, the other succeeds.

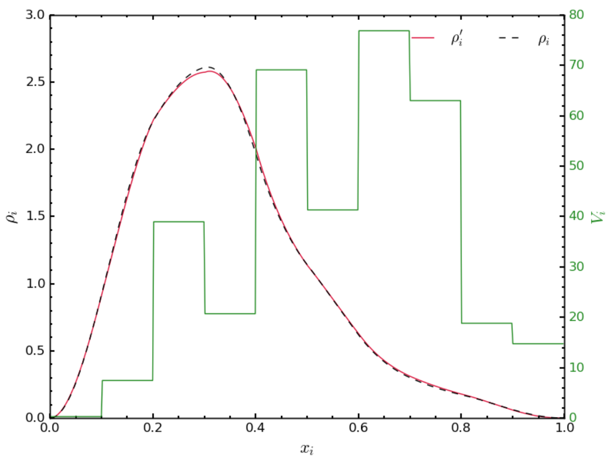
number of parameters in the recurrent cell) is forced to learn the underlying mapping. A consequence of the GE taking the entire potential profile as input is that the extended architecture no longer has spatial scalability.
Like in section 3.3 we could now proceed with training the model for different sizes of the hidden state to search for the most parsimonious representation where the loss is not compromised. However, the training of this model is sensitive due to the scale difference between output and input. Thus, training a new model from scratch for gradually fewer parameters is not a preferred method. Instead, we will attempt to use the RAE to find the minimal sufficient representation. To do that, the RAE will have to be extended like the RNN-solver was. Granted similar training issue will be present in this case for the RAE. The significant difference is that the RAE is here tasked with reproducing the hidden states h_i instead of producing the density ρ_i. We experience, that the machine has an easier time learning the minimal representation this way.
The entire extended learning algorithm is illustrated in fig. 37. Applying this, we will examine the information encoded in the global variable z.

Conclusion

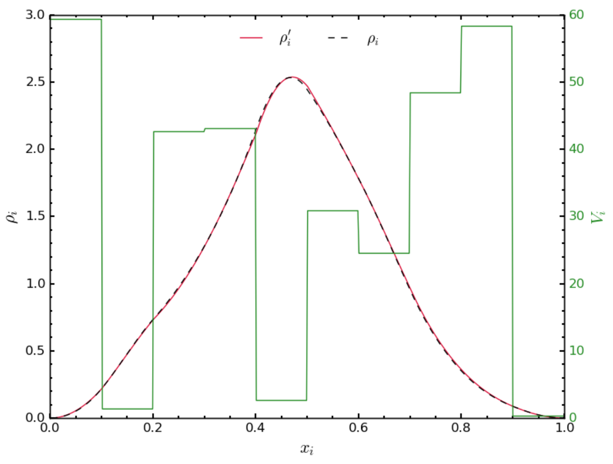
We have implemented a Global Encoder to extract necessary information about a non-local system. The GE+RNN model can predict the probability density ρ_i with $\mathcal{L}^{\rho_i}_{MSE} < 10^{-4}$. Additionally, we have developed an optimization method that allows us to train complex models on a difficult dataset.

4.4 Extended Recurrent Auto Encoder

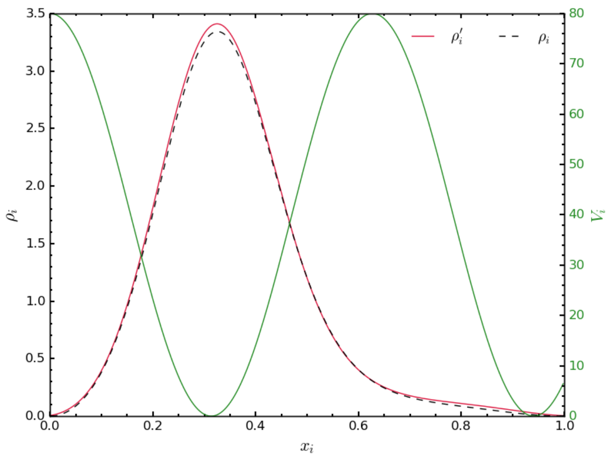
In this section we look to combine GE and RNN with the RAE introduced in last section. The assignment of the RAE is unchanged: To encode a hidden state $h_{i_0} \in \mathbb{R}^d$ from the RNN-solver at a certain step i_0 (not limited to h_0) and evolve the compressed hidden state $\tilde{h}_{i_0} \in \mathbb{R}^{\tilde{d}}$ in a smaller recurrent cell. After each update a decoder attempts to reconstruct the hidden cell h_i from the updated $\tilde{h}_i$ for $i = i_0 + 1, i_0 + 2, ..$ etc. To properly update the compressed hidden states, the global variable z from GE must be fed to the RAE recurrent cell. But it is



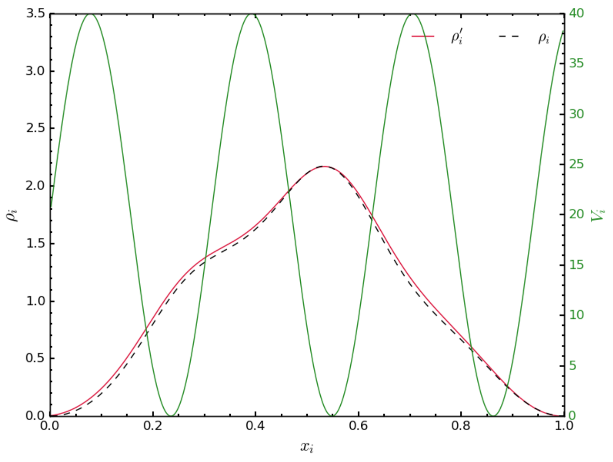
(a) Test Potential 1.



(b) Test Potential 2.



(c) Exponentially decaying periodic potential



(d) Periodic potential.

Figure 30: Extended RNN model: Predictions of different probability distributions for observing the quantum particle based on the potential profiles (in green). The prediction is compared to the numerically simulated target distributions. The first two plots are from test potentials that have not been trained on but constructed with the same method. The last two potential profiles are created differently from the data generation process to demonstrate the versatility of the model.

not enough to just provide z. If the RNN-solver is not the minimal and sufficient[19] representation, irrelevant information could be stored in the hidden variables. As a result the information in z might also need to be changed to fit the RAE model. Therefore, z is also passed trough an encoder (transformation). The z-encoder $\tilde{Z}$ is modelled by a 1-layer FCNN with

z-transformation:

$$\tilde{z} = \tilde{Z}(z) = W^{(\tilde{z})} * \mathrm{ELU}\left(W^{(h_1)} * z + b^{(h_1)}\right) + b^{(\tilde{z})} \tag{76}$$

Where $z \in \mathbb{R}^{d_z}$ and $\tilde{z} \in \mathbb{R}^{d_{\tilde{z}}}$. In this case $\tilde{Z}$ is not strictly an encoder, since we allow $d_z = d_{\tilde{z}}$. For example we have $d_z = 1$ in the extended RNN-solver above, which means $d_{\tilde{z}} = 1$. In that case $\tilde{Z}$ transforms the scalar z to another scalar $\tilde{z}$ more useful for the compressed RAE model.
We extend the recurrent cell of the RAE similarly to RNN-solver cell, utilizing an MTE update equation with the variables $\zeta = (V_i, \tilde{z}_1, .., \tilde{z}_{d_{\tilde{z}}})$

$$\tilde{h}_i = \widetilde{W}(V_i, \tilde{z}) * \tilde{h}_{i-1} \quad , \quad \widetilde{W}(V_i, \tilde{z}) = \sum_{0 \leq |n| \leq n_f} \widetilde{W}^{(n)} \zeta^n \tag{77}$$

n is again multi-index notation for $n = (n_V, m_1, .., m_{d_{\tilde{z}}})$ and n_f is the order of the MT expansion. The update equation is formulated for general applications where the appropriate n_f, and $d_{\tilde{z}}$ are not yet known. However, we have already found $n_f = 1$, $d_{\tilde{z}} = 1$ sufficient for the current system. The new RAE cell is shown in fig. 31b. Hidden state encoder and decoder are constructed in the same way as in section 3.4.

Encoder:

$$\tilde{h}_{i_0} = E(h_{i_0}) = W^{(o)} * \mathrm{ELU}(W^{(h_1)} * h_{i_0} + b^{(h_1)}) + b^{(o)} \tag{78}$$

Recurrent cell:

$$\tilde{h}_i = \widetilde{W}(V_i, \tilde{z}) * \tilde{h}_{i-1}, \quad (i = i_0 + 1, i_0 + 2, \cdots) \tag{79}$$

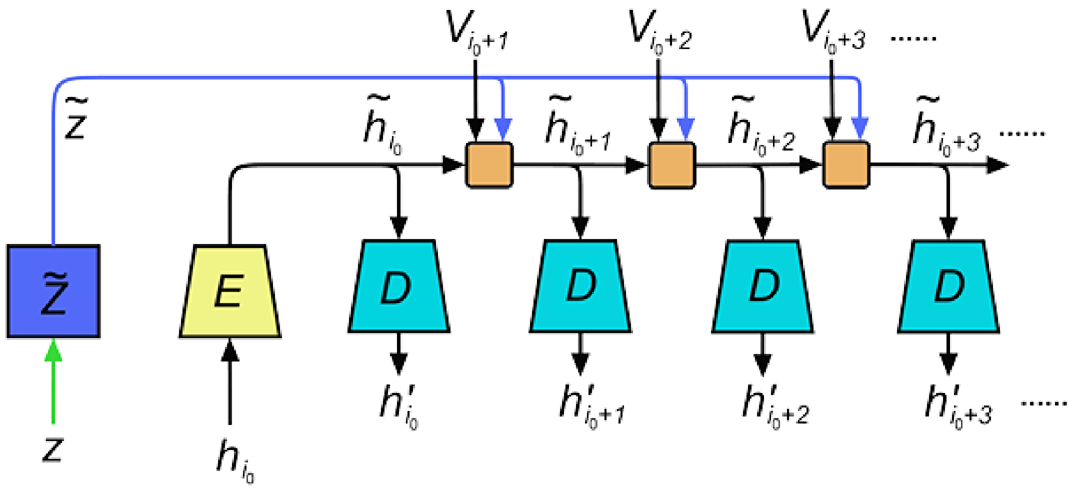
Decoder:

$$h'_i = D\left(\tilde{h}_i\right), \quad (i = i_0, i_0 + 1, i_0 + 2, \cdots) \tag{80}$$

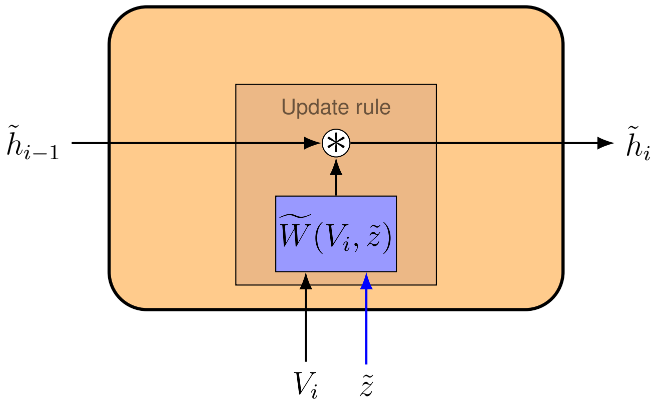
$\tilde{Z}$, E and D are constructed with 200 nodes in the hidden layer. We train the RAE based on its ability to reconstruct the different hidden states h_i with the loss function $\mathcal{L}^{h_i}_{MSE}$.
The flow of information for the ext. RAE model is illustrated in fig 31a.
The ext. RAE is trained for different dimensions of the compressed state $\tilde{d}$ to find the smallest possible representation. First the entire ext. RAE model goes through a coarse training. Adam is used as optimizer with $\tau = 2 \cdot 10^{-5}$. This training is (for unclear reasons) not very volatile for the RAE. The training can therefore be done without so called annealing, i.e. manually lowering τ (granted Adam already adjust the step-size throughout training). When the loss of the model is reduced to some acceptable level, in this case, $\mathcal{L}^{h_i}_{MSE} < 10^{-2}$. The model is further fine-tuned by alternating between adjusting only the $\tilde{Z}$, E and D parameters and the recurrent cell parameters for a set number of epochs.
In fig. 32 the coarse training phase for different $\tilde{d}$-values of the ext. RAE is shown. Like in the previous section, we find that for $\tilde{d} \geq 2$ the loss can be reduced to a similar degree but not for values lower than 2[20]. Therefore, we observe that the learning architecture correctly identifies the minimum number of variables needed, despite being provided with a more difficult non-local dataset. However, we have to be careful when concluding on the result of the training process of a complex model and/or difficult data. The likelihood, that a more parsimonious (but sufficient) representation exists, but simply fails to to converge in training, is much higher in this scenario.

By analysing the encoded variables of the model, we can determine if the model has discovered the underlying ground state wavefunction and the allowed energy. We first consider the global variable $\tilde{z}$. In fig. 33, The output of $\tilde{Z}$ for the $\tilde{d}$ model is plotted as a function of the ground state energies from the simulated test data.

[19]Minimal and sufficient means the smallest representation where the prediction loss is not sacrificed.
[20]Note that a model with $\tilde{d} < 2$ cannot be trained even when implementing the training method from the ext. RNN model.



(a) The structure of the extended RAE model. The global variable z and h_{i_0} is taken from a successfully trained extended RNN model. z is subjected to a transformation $\tilde{Z}$ to provide the needed information to the RAE cell.



(b) The RAE cell with a MTE update rule. Both local potential V_i and encoded global variable(s) $\tilde{z}$ are now given as input each step in the sequence. The updated hidden state h_i is a matrix product of MTE matrix $\tilde{W}(V_i, z)$ and h_{i-1}.

Figure 31

The variable $\tilde{z}$ appears to be very strongly correlated with the energy. Calculating the first-order correlation coefficient [25] between E and $\tilde{z}$, we find

$$\langle \tilde{z}E \rangle \frac{\overline{\tilde{z}E} - \bar{\tilde{z}}\bar{E}}{\sigma_{\tilde{z}}\sigma_E} = 0.9985... \approx 1$$

, where $\bar{X}$ is the mean value of the respective variables for the validation data. $\sigma_{\tilde{z}}$,σ_E is the standard deviation of $\tilde{z}$ and E, respectively. We conclude that the architecture, in this case completely unsupervised, chooses to store the energy of the quantum particle (possibly scaled due to free basis choice) as a global variable in order to predict ρ_i.
Additionally, we consider the activation of the global encoder z,(see fig. 33), here for the trained $d = 6$, $d_z = 1$ and $n_f = 1$ model. The same first-order correlation is calculated for z and E,

$$\langle zE \rangle = 0.704...$$

The correlation between the encoded information and the energy is far less apparent, which shows how reducing the model to a minimal representation forces the machine to get rid of all useless information.

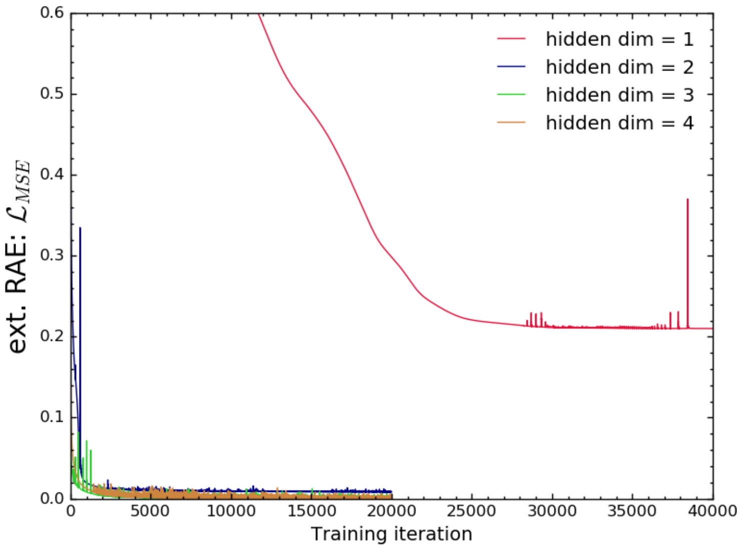


Figure 32: Coarse training phase for the extended RAE: $\mathcal{L}_{MSE}^{h_i}$ over the training period for different sizes of the compressed hidden states. The $\tilde{d} = 1$ model is trained for more iterations to show that the model never converges. It should also be noted that the loss for $\tilde{d} = 2$ is slightly higher than for the other models. Afterwards, meticulous fine-tuning is performed for the minimal representation to extract precise values for the physical properties of the DSE system.

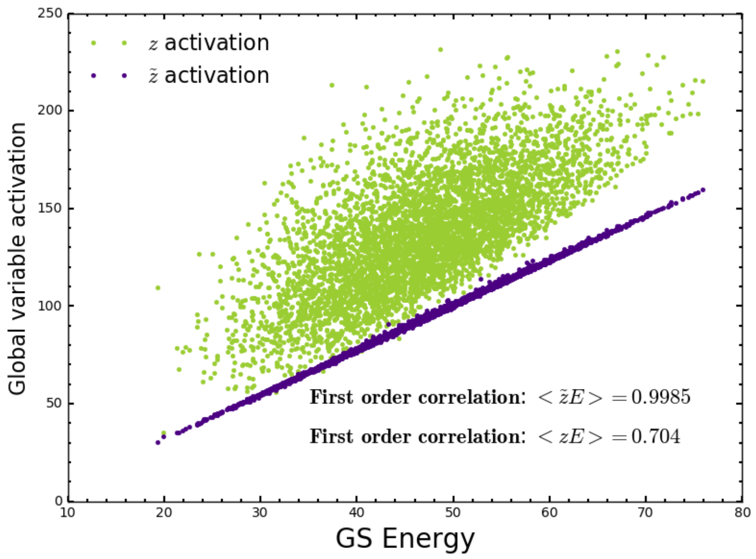


Figure 33: Plot of both RAE global variable $\tilde{z}$ and the z vs the allowed ground state energies E. E and $\tilde{z}$ are completely correlated unlike E and z, which are only weakly correlated. Shows that redundant information is present in the d=6 extended RNN model.

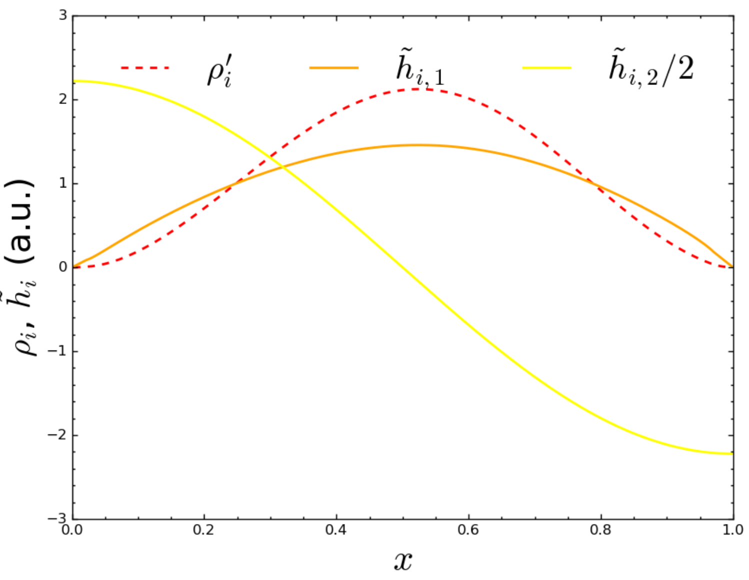


Figure 34: The 2 hidden variables $\tilde{h}_i$ for the ext. RAE are plotted with the ext. RNN prediction ρ_i for constant potential $V = 40$.

The variables stored in the latent vectors $\tilde{h}_i$ is compared to the wavefunction ψ_i and the forward difference quotient

$$\Delta\psi_i = \frac{\psi_{1+i} - \psi_i}{\Delta x}$$

Like in the previous section, we have to deal with the ambiguity of the hidden state basis. Fortunately, due to the linear configuration of the update law, all possible bases are related by a linear transformation. We can find a change of basis matrix M that can brings $\tilde{h}_i$ on the standard basis, which allows us to compare it to the simulated data. The mean relative error between ψ_i and $\tilde{h}_{1,i-1}$, denoted $\langle\Delta\rangle_{\psi_i \sim \tilde{h}_{1,i-1}}$, is calculated for the test samples. The same is done for $\Delta\psi_i$ and $\tilde{h}_{2,i-1}$

$$\langle\Delta\rangle_{\psi_i \sim \tilde{h}_{1,i-1}} \simeq 2.8\% \quad , \quad \langle\Delta\rangle_{\Delta\psi_i \sim \tilde{h}_{2,i-1}} \simeq 4.2\%$$

The i^{th} wavefunction is compared to the $(i-1)^{th}$ hidden state due to the cell setup. The relative error, being less that 5 percent, is solid evidence that the RAE has stored the wavefunction and its difference quotient (derivative) in $\tilde{h}_i = (\psi_{i+1}, \Delta\psi_{i+1})^T$ in order to reconstruct h_i.

In fig. 34, the compressed latent state $\tilde{h}_i$ (+ basis change) and ρ_i for constant potential $(V = 40)$ is plotted. For this simple case we know that $\psi(x) = \sqrt{2}\sin(\pi x)$ and $\partial_x\psi(x) = \pi\sqrt{2}\cos(\pi x)$. We observe, that the values stored in $\tilde{h}_i$ agrees with the expected values.

Knowing that ψ_i and $\Delta\psi_i$ are stored in the latent variables and that the $\tilde{Z}$ activation is proportional to the energy, strongly indicates that the update equation has formulated some version of the DSE. We confirm this by looking at the weight coefficient matrices in $W(V_i, z)$.

$$M^{-1}\tilde{W}^{(0,0)}M = \begin{pmatrix} 1 + 3\cdot 10^{-7} & 0.001994 \\ 45\cdot 10^{-7} & 1 - 25\cdot 10^{-7} \end{pmatrix} \approx \begin{pmatrix} 1 & \Delta x \\ 0 & 1 \end{pmatrix} \tag{81}$$

$$M^{-1}\tilde{W}^{(1,0)}M = \begin{pmatrix} 55\cdot 10^{-7} & -7\cdot 10^{-7} \\ -0.001997 & -41\cdot 10^{-7} \end{pmatrix} \approx \begin{pmatrix} 0 & 0 \\ -\Delta x & 0 \end{pmatrix} \tag{82}$$

$$M^{-1}\tilde{W}^{(0,1)}M = \begin{pmatrix} 2\cdot 10^{-6} & 76\cdot 10^{-7} \\ 0.002000 & -9\cdot 10^{-7} \end{pmatrix} \approx \begin{pmatrix} 0 & 0 \\ \Delta x & 0 \end{pmatrix} \tag{83}$$

Hence, the update equation becomes: [21]

$$\tilde{h}_i = \left(\widetilde{W}^{(0,0)} + \widetilde{W}^{(1,0)}V_i + \widetilde{W}^{(0,1)}\tilde{z}\right) * \tilde{h}_{i-1} = \begin{pmatrix} 1 & \Delta x \\ \Delta x(\tilde{z} - V_i) & 1 \end{pmatrix} * \tilde{h}_{i-1} \tag{84}$$

We see that a discrete version of the Schrödinger equation emerges from the minimal sufficient representation of the recurrent learning architecture. One important thing to note on the nummerical implementation. The precision of the weights and variables needs to be higher for the neural model to learn the important features of the infinite boundary condition GS dataset. In the previous section, PyTorch 32-bit floating point precision was enough, but now 64-bit double precision is needed. This results in much longer training times.

Conclusion

The RAE architecture can from the latent variables of the extended RNN model discover the discrete Schrodinger equations and as well as identify the energy as the essential (global) variable needed to do the the local update of the hidden states.
The complexity of the many factors involved in assembling and tuning this "physics machine", makes this a noteworthy result.

4.4.1 Independence of Global Encoder and RNN-solver

The learning architecture is no longer scalable. However, that is only partially true, since in the current setup only the GE needs to be fed the entire potential sequence. It should therefore be possible to retrain the GE with a new system length L, but leave the RNN-solver unchanged. The GE needs to provide the global variable(s) such that the basis of the RNN-solver (chosen at the prior training) can still solve the system. This can be done by freezing the parameters of the RNN-solver such that only the GE parameters are tuned. We can then train the GE on a different system length. To do that we create 10000 samples with $L = 0.5$. Only training the GE greatly simplifies the training and a good performing model can be obtained with default settings.
The retrained GE + RNN model is evaluated with a 3000 sample validation set. We find mean relative prediction error to be 2.4%, demonstrating that the model predicts the probability density with similar relative error less than 5%. The grid spacing Δx must be kept constant under the different sequence lengths, otherwise the RNN-solver must be retrained as well. Therefore, increasing the system length simply means adding more points to the grid.
In fig. 35 predictions for the retrained model is showcased. An important point is, that the quality of the first system length training heavily impacts the prediction accuracy of the retrained GE model. If the RNN cell parameters have not been completely trained, the loss of the retrained model will be amplified.

The independence of the GE and RNN can be exploited even further. Using the same procedure, it should be possible to retrain the GE for different boundary conditions (periodic, Dirichlet etc.) as well. We generate data with periodic boundary conditions to test this. Numerical solutions to the SE with periodic boundary conditions can be found by imposing $\psi_0 = \psi_{N+1}$, which results in a slight modifications to the Hamiltonian. $[H]$ becomes a $(N+1) \times (N+1)$- matrix[22] with the two new non-zero elements $H_{0,N} = H_{N,0} = -t_0$. Subsequently, 10000 samples are generated using the method described in section 4.1.
We then proceed with training the GE on pairs of $(\{V_i\}, \{\rho_i\})$ generated from periodic boundary conditions where RNN cell parameter values are frozen after training on infinite potential boundary data. This is done for the $d = 5$, $d_z = 1$ and $n_f = 1$ cell.
In fig. 36 density predictions for 2 potential sequences not used in the training is shown. The relative error on a 3000 sample validation set is found to be $\langle\Delta\rangle_{\text{periodic}} = 3.3\%$, showing that the recurrent cell can be reused and still provide accurate density predictions.

[21] For illustrative purposes the similarity transformation is not shown.

[22] We could also have it be a $N \times N$-matrix but that would result in $\Delta x = N/L$. In order to keep the grid spacing constant we add the 0^{th} grid point to the matrix. The single grid point makes little difference in practise.

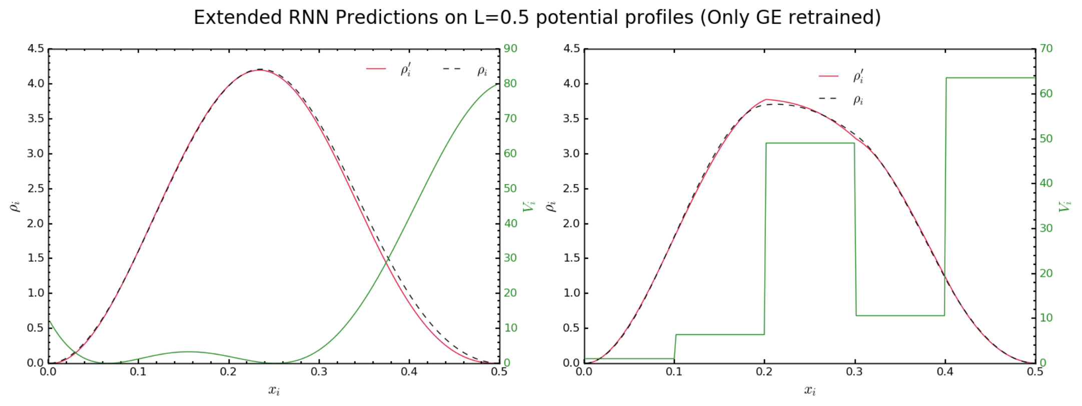


Figure 35: Comparison of target density and output from the RNN extended with GE retrained for a different length. Left) Potential sequence different from the generated dataset. Right) Arbitrary test set potential.

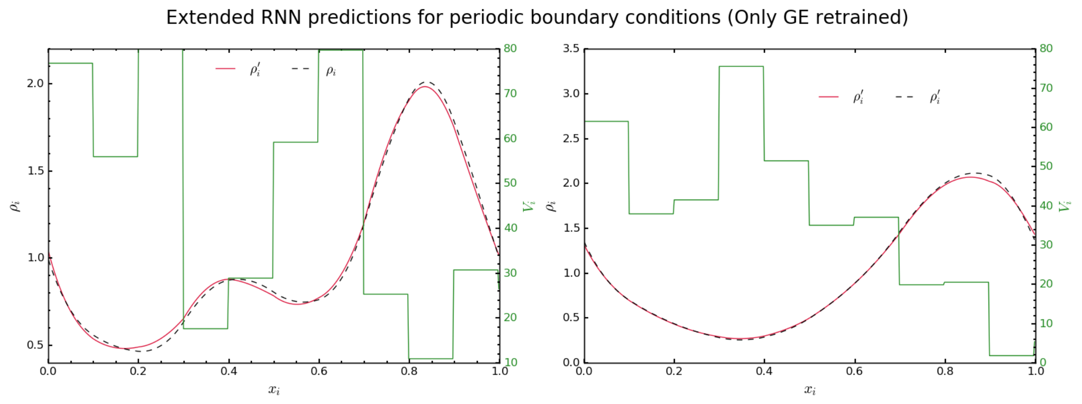


Figure 36: Comparison of target density and output from the RNN extended with GE retrained on generated data with periodic boundary conditions. Generally, slightly worse predictions than the original dataset since the smallest inaccuracy in the parameter values for the recurrent cell will lead to amplified prediction errors.

Thus, while the entire model is not scalable, we have demonstrated that it can be separated into a local and non-local component, which can be trained separately. We envision, this will greatly increase the usefulness of the architecture.

4.4.2 Higher Energy Levels

In this section, we have only considered ground state solutions. But we imagine generalizing to higher excitation levels to be rather straight forward. Different energy levels cannot be trained at the same time, since ground states and excited states are solutions to distinct allowed energies for the *same* potential. If the neural model is simply trained on a dataset with states of mixed energy levels, it will fail majestically. The model needs information specifying, which energy level is given.
This could potentially be done by feeding the global encoder an additional number Λ, to help specify the energy level. $\Lambda \in 0, 1, 2..$ corresponding to the excitation level. This would result in the input of the GE being an (N+1) vector. Alternatively, if that is not sufficient, the model could be expanded with multiple GEs, one for each excitation. Λ would then decide, which GE the potential is fed to.

4.5 Discussion

In this section, we expand the recurrent learning architecture with the addition of a global encoder. The new component allows the combined neural model to handle systems where information about the entire input se-

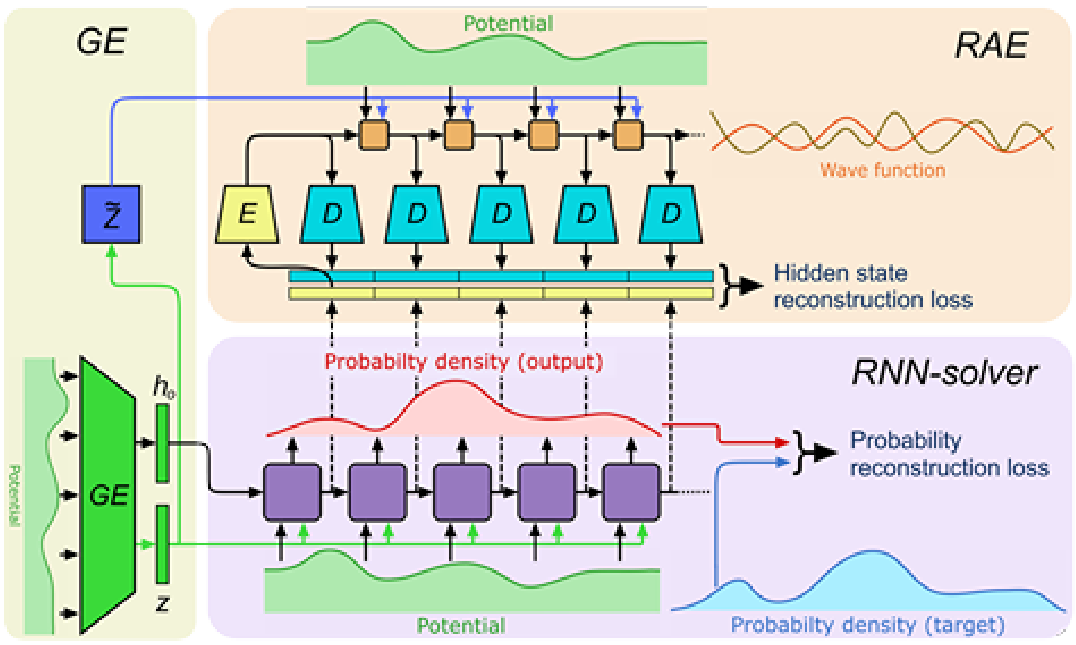

GE
RAE
Potential
Wave function
$\tilde{Z}$
E
D
Hidden state reconstruction loss
Probabilty density (output)
RNN-solver
Probability reconstruction loss
h_0
GE
z
Potential
Probabilty density (target)

Figure 37: Extended version of the learning machine. The architecture consist of three major components. The global encoder (left) tasked with learning variables, which require the entire input sequence. The RNN-solver (lower) is tasked with solving the potential-to-density mapping with minimal loss. The global variable(s) z (green line) is provided as additional input at each step. The RAE (upper) is implemented to compress the information contained in the RNN to extract the important features of the system. $\tilde{z}$ (blue line) is given as additional input each time the compressed hidden state is updated. The z-transformer allows the prediction task and knowledge distillation to remain independent tasks.

quence is needed. The complete learning architecture is shown in fig. 37. The 3 major components (GE, RNN and RAE) each perform a different task. The RNN-solver learns to perform the potential-to-density mapping, the RAE is tasked with distilling information from the hidden states h_i and the GE is an auxiliary component, that stores essential information from the entire input sequence.
We emphasize, that even with the addition of the GE, the prediction and information extraction are still completely separate tasks. The training of the RAE does not influence the RNN prediction of ρ_i in any way. The z-transformer $\tilde{Z}$ plays a key role in keeping the two tasks separate. If we removed $\tilde{Z}$, there would likely be a competition between the RNN and RAE about, which values would be stored in z. Implementing $\tilde{Z}$ avoids this potential conflict. It allows for GE to be trained only based on obtaining the most accurate predictions, because $\tilde{Z}$ can be optimized to extract the relevant information from z.

To prepare for the prospective use of the network architecture on other systems, many aspects have to be kept in mind. Model optimization/training is central to consider, when attempting to retain all information about the models solution method. RNN models are inherently difficult to train; when paired with an unbalanced dataset the difficulty increases even further. Since smoothing of the loss surface through techniques like batch normalization is not an option, realizing successful models is no easy task. Thus, we consider it no small accomplishment to have actually optimized a model, that extract and conserve all essential information about the physical system.
However, the single particle Schrödinger equation is a simple quantum system. The extraction process will only be more uncomfortale as the complexity of the system increases. It is entirely possible that models for such systems cannot be trained. We imagine that, at least, new optimization methods will have to be developed for this to be possible.

Additionally, the effectiveness of the multi-variable Taylor expanded update law must be discussed. The MTE update equation differs from the previous Taylor expansion rule. The cell no longer only has a polynomial

dependency on the input. The update equation now has a non-trivial non-linear dependency on the the entire input sequence through z.
However, the primary limitation of the Taylor update equation remains. For the update equation to approximate complicated functions, the order of expansion will have to be high. Meanwhile, the advantage of the Taylor-expansion, its analytical tractability, is lost when the higher-order expansion terms are included. In that case, an FCNN might as well be used as the update rule of the cell, since the local input-dependency is impossible to gauge regardless. The MTE update rule is only effective when the update function can be modelled effectively by first- or second-order input expansions.
To improve the predictions, we can instead think of the neural model as a numerical solution identifier [26]. Rather than using a high-order MTE update law, we can decrease the local truncation error by taking more points into consideration when estimating the slope/tangent line in each step in the sequence. The local truncation error of numerical integration refers to the deviation from the exact solution, caused by a single step. The local truncation error of a first-order numerical method is proportional to the square of the step-size (Δx in this system). A second-order method has a local truncation error proportional to the third power of the step-size and so on. The solution, stored in the minimal representation of the ext. RAE, is a first-order numerical solution (Euler integration). We speculate, that increasing the size of the hidden state would be equivalent to allowing the neural model to discover higher-order numerical solution methods. This is, however, difficult to verify, since increasing d also means a higher probability of redundant information being stored in the hidden states.
It would require a high-order numerical solution to accurately predict the output sequence. This is impossible to test for this particular system. since the simulated dataset is, in fact, a first-order numerical solution of the SE equation in the first place.
On the other hand, it could be interesting to test for the dataset of the previous section. Gradually increasing a should at some point necessitate an increase in the hidden state dimension for the predictions of the model to remain accurate. An attempt of this was illustrated in fig. 21 for an $a = 2$ dataset. Increasing d did not seem to matter in that case, but that could be because $d > 8$ was needed for the $a = 2$ step-size. Another possibility is that the RNN-models was not completely optimized during the training process. It could be interesting to test this for less extreme values of a.

The numerically simulated data, used in this section, is another interesting point of discussion. So far, we have yet to consider whether the DSE solutions are even good approximations to the solutions of Eq.(67). The quality of the numerical solutions is completely irrelevant to the neural model. The neural architecture learns a variation of the DSE to solve the system, independently of how good the DSE approximates the continuous real-space SE. We see that from having discovered, that construction of difference (recurrent) equations is how the neural architecture attempts to solve systems.
However, being aware of how the neural model tries to solve problems is of immense importance pertaining to real-world application of the architecture. Consider an experimental setup that perfectly mirrors the system of this section[23]. Observing the particle (collapsing the wavefunction repeatedly) allows us to gather different probability distributions of the particle position. Assuming that the data gathering process is completely accurate, the distributions we obtain, are **exact** normalized solutions to the SE. The neural model, however, does not realize this difference. The solution method of the model will be the same as for the numerically simulated dataset. Whether the model can capture the governing equations of the system, depends on how well a numerical solution can model the observed output sequences[24]. The quality of a numerical solution can be made arbitrarily good by decreasing the spacing between points enough (The Local truncation error is always proportional to some power of the grid-spacing. The power of depends on the order of the numerical solution method, but even for first-order methods the error $\propto \Delta x^2$ and hence goes to 0 for $\Delta x \to 0$). Therefore, We can in principle always provide the model with a dataset where the distance between datapoints is small enough that even first-order numerical solutions are accurate approximations. However, at some point, the grid-spacing will have to be

[23]E.g. some subatomic particle trapped in a cavity, where we can regulate the potential.

[24]The training procedure will, of course, vary in difficulty depending on the dataset. But in this part of the discussion, we assume that any valid numerical solution can be trained.

so small that the computers nummerical precision on model parameters and variables becomes a limiting factor. Increasing the latent state of the recurrent model would allow one to reduce the need for small step-sizes by capturing higher-order numerical solutions. However, it is desirable to keep the hidden state as small as possible. Finding the minimal sufficient model is how we extract meaningful information about the underlying system. If the dimension of the hidden state, that needs to be increased, passed a certain point (for both RNN and RAE), we lose the ability to extract information from the hidden state. In that scenario, a universal update rule modelled by an FCNN is to be preferred. However, an FCNN update rule comes with other problems, which will be addressed in the next section.

In regards to making scientific discoveries using this type of architecture, it is highly idealistic to assume that the provided data is perfectly tailored to the learning architecture. The unknown data could look like the $a = 2$ data from the previous section, where the learning architecture is simply not effective. Thus, it is not realistic to expect that any existing single variable 1-D mapping can be learned by the model. However, the number of feasible applications seems to be greater than first assumed in section 3.5.
An obvious problem, that architecture should be able to solve, is the time-evolution of the stationary eigenstates found in this section. The architecture should be able to learn the first-order differential equation without too many complications.
Regrettably, the learning architecture is limited to single variable and 1-D systems. We did not manage to conceptualize a viable multivariable/higher-dimensional version of the learning architecture. This would be an excellent quest to explore in future works. A multivariable/higher-dimensional recurrent architecture would open up for a large number of applications in the condensed matter field.

In order to retain the solution method, we have had to sacrifice a part of what makes neural networks effective. Some parts of the black box must remain closed for neural networks to remain useful for more complicated physical systems (Many-body-problems, higher-dimensional systems etc.). This is leading us to the conclusion that an alternative approach to the concept of scientific discoveries through AI might have to be pursued.

5 Extended State Scattering

In this section, we consider a freely moving particle scattering from a potential barrier. The transmission probability T can be determined based on the potential V and particle energy E. The dataset $(\{E, V\}, T)$ is ill-suited for the learning architecture with a linear (in the hidden state) update law. As a result, a different recurrent model is designed. A key focus in this section is to outline the limitation of information extraction from RNNs with non-linear update laws.

5.1 Physical System

A transfer-matrix approach is used to determine the transmission probability T (commonly called the transmission coefficient) for a particle passing through a series of rectangular potential barriers. Depending on the barrier width any 1D-dimensional potential can be approximated by rectangular barriers to arbitrary accuracy making this approach suitable for any 1D quantum system. We mirror the approach from Ref. [27] in setting up the formalism. The starting point, again, is Eq.(67). For constant $V(x)$ a general solution to $\psi(x)$ is a linear combination of the left and right moving free particle wavefunction:

$$\psi(x) = A\,\mathrm{e}^{ikx} + B\,\mathrm{e}^{-ikx}, \tag{85}$$

where $k = C\sqrt{E - V}$ (dimensionless SE) and C is a constant to used to alter the general strength of the rectangular barriers. Using a matrixproduct formalism, the wavefunction can be expressed as the following dot product

$$\psi(x) = \begin{pmatrix} e^{ikx} & e^{-ikx} \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} e^{ikx} & e^{-ikx} \end{pmatrix} \varphi, \tag{86}$$

where $\varphi = \begin{pmatrix} A, & B \end{pmatrix}^T$ is a coefficient vector.
Next we consider a translation of the position basis by some length λ such that $x' = x - \lambda$. Particularly, λ is always chosen such that both coordinates lies within the same rectangular barrier (i.e constant potential). The same wavefunction can be expressed using both coordinates i.e. $\psi(x) = \psi'(x')$, hence

$$\psi'(x') = \psi(x' + \lambda) \Rightarrow$$

$$\psi'(x') = \begin{pmatrix} e^{ikx'} & e^{-ikx'} \end{pmatrix} \varphi' = \begin{pmatrix} e^{ik(x'+\lambda)} & e^{-ik(x'+\lambda)} \end{pmatrix} \varphi = \begin{pmatrix} e^{ikx'} & e^{-ikx'} \end{pmatrix} \begin{pmatrix} Ae^{ik\lambda} \\ Be^{-ik\lambda} \end{pmatrix},$$

with $\varphi' = \begin{pmatrix} A', & B' \end{pmatrix}^T$. From this we see $A' = Ae^{ik\lambda}$ and $B' = Be^{-ik\lambda}$. The change of the coefficient under a translation can be represented by a diagonal propagation matrix,

$$p_i = \begin{pmatrix} e^{ik_i\lambda} & 0 \\ 0 & e^{-ik_i\lambda} \end{pmatrix}, \tag{87}$$

where $k_i = C\sqrt{E - V_i}$ and V_i is the height of the i^{th} barrier. Next, we have to deal with the discontinuous jump from one rectangular barrier to another. Use of the propagation matrix with λ as the barriers width, means the discontinuities between two barriers are always placed at $x = 0$. Lets consider the discontinuity between the i^{th} and the $(i+1)^{th}$ barrier. Here $V(x) = V_i$ for $x < 0$ and $V(x) = V_{i+1}$ for $x > 0$. Hence, $\psi(x)$ is given by:

$$\psi_i(x) = A_i e^{ik_i x} + B_i e^{-ik_i x} \qquad x \leq 0$$

$$\psi_{i+1} = A_{i+1} e^{ik_{i+1} x} + B_{i+1} e^{-ik_{i+1} x} \qquad x > 0$$

A relation between the coefficients is derived by requiring the wavefunction and its derivative to be continuous at the potential (non-infinite) discontinuity ($x = 0$). These conditions yield:

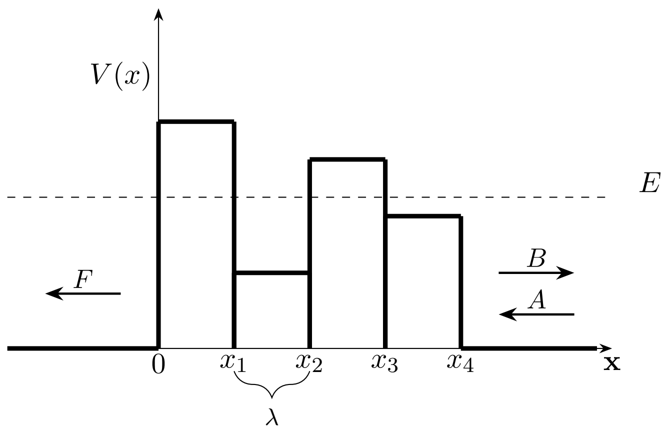


Figure 38: Scattering from a potential consisting of 4 rectangular barriers.

$$A_i + B_i = A_{i+1} + B_{i+1},$$
$$A_i + B_i = k_{i+1}/k_i \left(A_{i+1} + B_{i+1}\right).$$

Subsequently, the equations can be solved to find recurrent relations for the coefficients. Using the coefficient vectors $\varphi_i = \begin{pmatrix} A_i, & B_i \end{pmatrix}^T$, the relations can be expressed as

$$\varphi_{i+1} = b_{i,i+1}\varphi_i = \frac{1}{2}\begin{pmatrix} 1 + k_1/k_2 & 1 - k_1/k_2 \\ 1 - k_1/k_2 & 1 + k_1/k_2 \end{pmatrix}\begin{pmatrix} A_i \\ B_i \end{pmatrix}, \tag{88}$$

with $b_{i,i+1}$ being the discontinuity matrix connecting the two barriers.
The p and b-matrices allows us to build a transfer matrices for any number of potential barriers. For a set of N_V barriers. Given a coefficient vector at the left of the barriers, φ_{left}, the coefficient vector at the right, φ_{right}, can be expressed as a combination of p and b matrices:

$$\begin{aligned} \varphi_{\text{right}} &= \left[p_{N_V} \cdots * p_{i+1} * b_{i,i+1} p_i \cdots * p_1 * b_{0,1}\right]\varphi_{\text{left}} \\ &= t\varphi_{\text{left}} \end{aligned} \tag{89}$$

Where t is the transfer matrix. The formalism can be used to effectively determine the probability of a particle traveling through a sequence of rectangular barriers $V(x) = \{V_i\}$.
Consider a particle incident on a potential from the right, as illustrated fig. 38. Upon hitting the barrier a reflected and a transmitted wave is created. In the coefficient vector, φ_{right}, the amplitude of the incident and reflected wave is made up by A and B, respectively. The transmitted wave on the left side of the potential only has a left moving wave, written as $\varphi_{\text{left}} = (0, F)^T$. Using t φ_{right} is given by

$$\varphi_{\text{right}} = t\varphi_{\text{left}} = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}\begin{pmatrix} 0 \\ F \end{pmatrix} = \begin{pmatrix} t_{12}F \\ t_{22}F \end{pmatrix}. \tag{90}$$

The probability of transmission is the ratio between the transmitted and incident wave, defined as the transmission coefficient [18]:

$$T \equiv \frac{|F|^2}{|A|^2} = |1/t_{2,2}|^2 \tag{91}$$

Notice that the amplitude of the incident wave cancels such that transmission coefficients can readily be generated based on the energy level of the particle and potential of the system.

We note that the method of connecting the wavefunctions at the discontinuities is similar to the procedure from section3. The matrix-product of p and b matrices actually yield the conditions used to find Eq.(27) and Eq.(28) (with a possible phase difference). However, the transfer matrix approach proves to be more efficient computationally.

5.1.1 Data Generation

Simulating the transmission coefficient data is different from the generation process in the two other systems. In section 4 the bound ground states of the DSE have specific quantized energies based on the potential. Here, we select an extended state, based on its energy, as the incident wave on the potential sequence. Next, the potential barriers are constructed. We choose the same width for all barriers. The length of potential is given by $L = N_V\lambda$, where N_V is the number of rectangular potential barriers and λ is the barrier width. $x_i = i\lambda$ is the position where the i^{th} barrier ends. The height/amplitude of the barriers are randomly decided between some values V_{min} and V_{max}.

Three different types of data can be generated. First, 'classical' transmission where the energy of the particle is always greater than the potential. Second, (nonclassical) quantum tunneling where the energy of the particle is always less than the height of the barriers. Third, mixed transmission where the V_i height can take values both higher and lower than the particle energy. For each potential, T is calculated for N_E linearly spaced energies, E, starting from the value E_{min} to E_{max}. The process of the data generation can be summarized in the following steps.

- Draw N_V discrete potential values V_i from a uniform distribution with $V_i \in [V_{\text{min}}, V_{\text{max}}]$.
- Create linearly spaced energy-vector with N_E elements, start value E_{min} and final value E_{max}. Depending on the type of data (classical, tunneling or mixed) the values of V_{min}, V_{max}, E_{min} and E_{max} change.
- $k_i = C\sqrt{E - V_i}$ are determined for each barrier and all energies. C is a constant to alter the general strength of the barriers.
- Discontinuity matrices $b_{i,i+1}$ and propagation matrices p_i are constructed
- The transfer matrices t are calculated from matrix multiplication of $b_{i,i+1}$ and p_i.
- Finally, $T = |1/t_{2,2}|^2$ is calculated.

The training dataset consists of 600 mixed sequences, 200 classical and tunneling sequences. This is repeated for 4 different number of barriers N_V (same λ), which results in the total sample size $N = 4000N_E$. Constants used to generate transmission coefficients for a 11-barrier potential ($N_V = 11$) is shown in table. 3. The constants used for the other N_V values are listed in appendix C1.
Additionally, a validation dataset of $N_{test} = 1000N_E$ samples with different barrier length N_V values is generated their respective constants can also be found in appendix C1.

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E	C
Mixed								
	11	0.05	25	0	20	2	50	2
Classical								
	11	0.05	9.999	0	20	10	50	2
Tunneling								
	11	0.05	30	10.001	10	5	30	2

Table 3: Parameters for the transmission coefficient data generation with 11 barriers i.e. $N_V = 11$.

WKB	λ	V_{min}	V_{min}	E_{max}	E_{min}	N_E	C
	0.05	11	14	10	2	9	2

Table 4: Constants used in when generating the WKB approximation dataset.

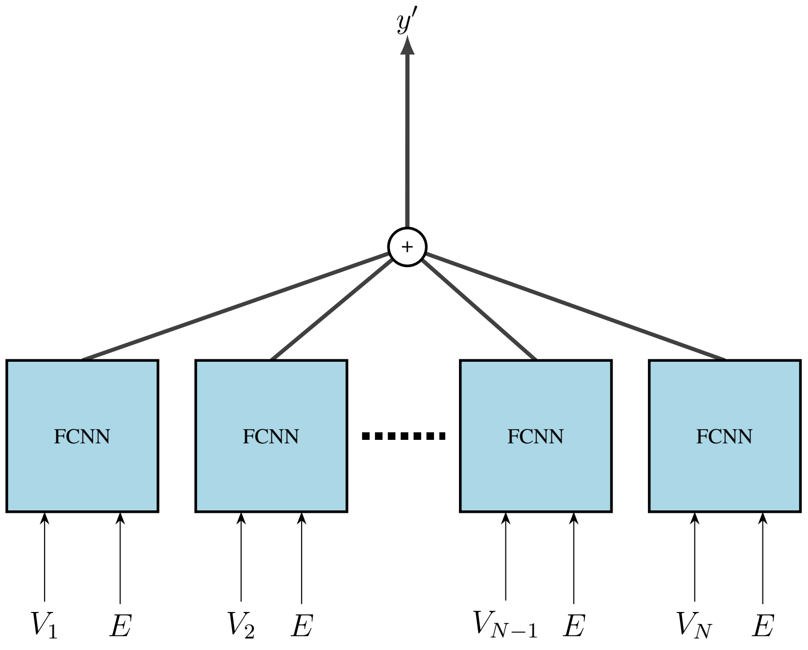


Figure 39: Illustration of the feedforward WKBnet. Energy and barrier height V_i is fed to each of the FCNNs. The output y' is the sum of individual FCNNs activation.

We, also, construct a smaller dataset with about 10000 samples of varying barrier length ($N_V = [5, 6, 8, 7, 9, 10, 11, 12, 13]$) where $V > E$ (see, other parameters listed in table.4). The potentials are constructed such that the WKB (Wentzel–Kramers–Brillouin) method should provide reasonable approximations of the tunneling probabilities. Here, smoothing of the randomly chosen V_i values is needed.

5.2 WKB Approximation

A fully-connected network could, as for the energy in the last section, be implemented to find the transmission probability. However, we take a different approach to explore another even simpler neural network model. We want to examine, if a network can emulate the WKB method of calculating tunneling coefficients. In non-classical regions ($E < V$), where the WKB approximation is valid, transmission coefficients are given by the equation:

$$T = \mathrm{e}^{-2\gamma}, \quad \text{where} \quad \gamma = \int_0^L |k(x)| \, \mathrm{d}x \tag{92}$$

A derivation of this is shown in appendix C2. From Eq.(92) it follows that

$$\ln T = -2 \int_0^L |k(x)| \, \mathrm{d}x, \tag{93}$$

where for sufficiently small barrier width, λ, the integral can be replaced by a sum.

$$\ln T = -2 \sum_{N_V} |k(x)| \lambda, \tag{94}$$

giving us a discrete method to determine $\ln T$.
Inspired by the WKB method, we design a feed-forward (**not** fully-connected) model, called WKBnet. The model is constructed as N_V parallel single-layer FCNN-cells. Each cell takes the particle energy, E, and a barrier amplitude V_i as input. Based on the input, each cell produces a scalar a_i as output. The output of the entire model is the sum of all cell outputs. The general structure of WKBnet is illustrated in fig. 39. WKBnet

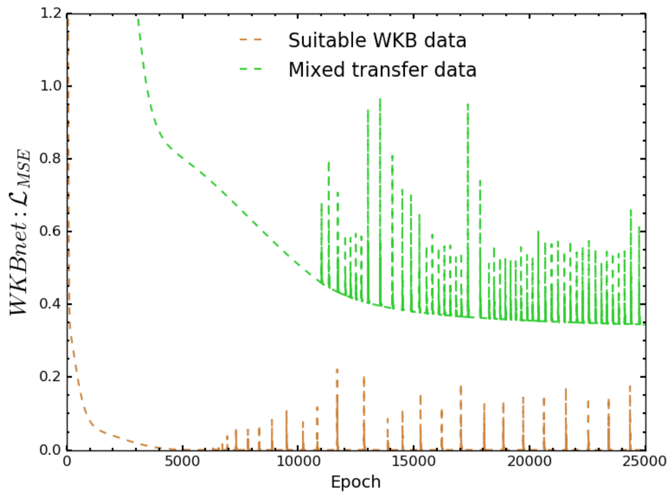


Figure 40: Training history for WKBnet on two different datasets. Both datasets consist of barrier-potentials but one generated such that WKB is likely a valid method to predict tunneling transmission coefficients. The second is the mixed transmission dataset, where WKB should be a poor approximation. An interesting phenomenon is the random spikes in loss throughout the training. The spikes could potentially be explained by the repeated use of the same FCNN-cell, which could make the loss unusually sensitive to certain changes in the parameters.

can be expressed algebraically with the following equations:

WKBnet:

FCNN-cell:

$$a_i = W^{(a)} * \text{ELU}\left(W^{(H_1)} * [V_i, E] + b^{(H_1)}\right) + b^{(a)} \tag{95}$$

Output:

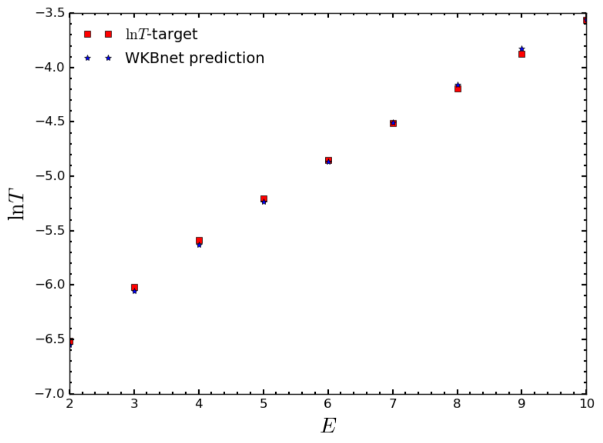
$$y' = \sum_{i}^{N_V} a_i \tag{96}$$

We train WKBnet to minimize, $\ln T$ and WKBnet's estimate of $\ln T$ is denoted $y' \in \mathbb{R}$. The training parameters are all contained in the FCNN-cell, the weight matrices $W^{(H_1)} \in \mathbb{R}^{d_H \times 2}$, $W^{(a)} \in \mathbb{R}^{1 \times d_H}$ and bias vectors $b^{(H_1)} \in \mathbb{R}^{d_H}$, $b^{(a)} \in \mathbb{R}$. d_H is the dimension of the hidden layer in the FCNN-cell. d_H is the only hyperparameter, concerning model construction, which highlights the simplicity of the model.
Note that the same FCNN-cell (i.e. same adjustable parameters) are utilized for each $[E, V_i]$-input. The advantage of applying the same FCNN-cell is that WKBnet becomes a scalable model that, once trained, should be able to predict $\ln T$ for potential sequences with a different number of rectangular barriers. However, this necessitates that all barriers have the same width, λ.
We expect that WKBnet should yield accurate predictions when trained on a dataset $(\{E, V\}, \ln T)$, where the WKB approximation is valid. We will, therefore, train the model on two different datasets. The WKB-tailored dataset and the mixed dataset where the approximation should fail. WKBnet is trained with the MSE loss function:

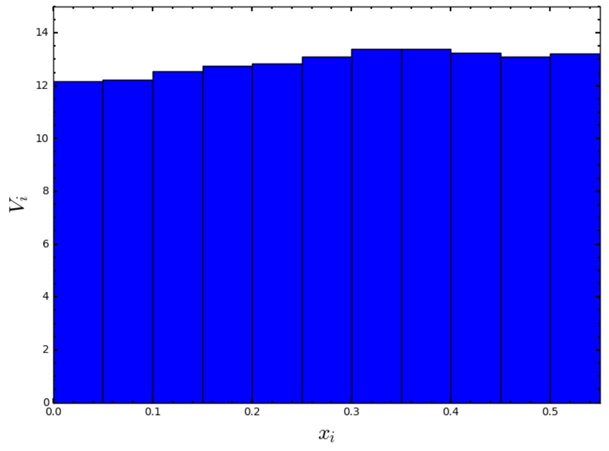
$$\mathcal{L}_{MSE}^{\ln T} = \sum_{i \in \text{batch}} (y' - \ln T)^2 \tag{97}$$

We construct a WKBnet model with $d_H = 200$. The model is trained on both datasets with the Adam optimizer, $\tau = 1 \cdot 10^{-4}$ and a batch size of 50 samples. The training history for the 2 datasets is illustrated in fig. 40. The loss is significantly lower when trained on the WKB-valid data. We also compute the mean relative error $\langle\Delta\rangle$ between the target and prediction data,

$$\langle\Delta\rangle_{\text{mixed}} = 164\% \quad , \quad \langle\Delta\rangle_{\text{WKB}} = 0.5\%.$$



(a) WKBnet predictions for different energies and 11-barrier potential.

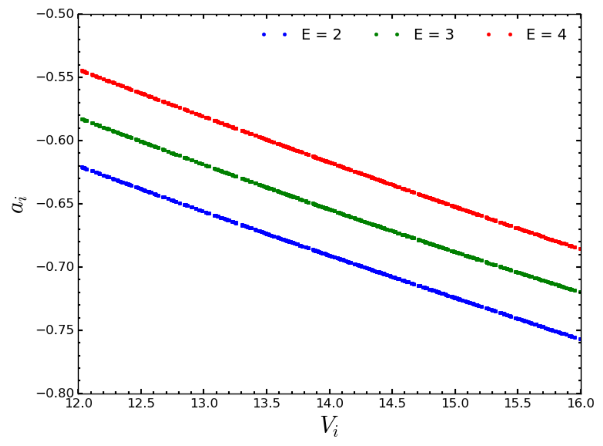


(b) Test potential from the WKB tailored dataset with 11 barriers.

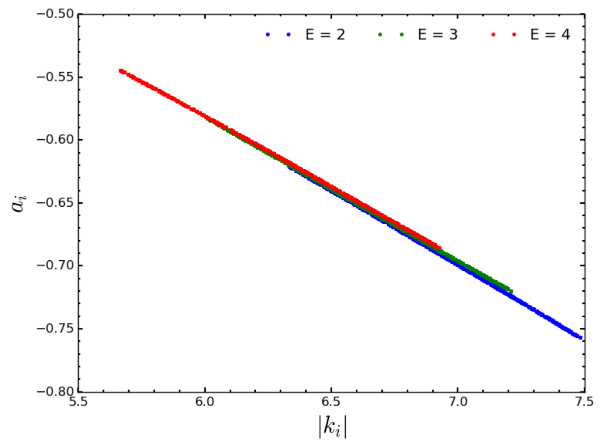
Figure 41: WKBnet : Predictions of the natural log of the transmission probability. a) Plot showcases the $\ln T$ prediction vs energy for a test potential ($N_V = 11$) constructed like the WKB dataset. b) Plot shows the test potential used to make the prediction in a).

The model fails to accurately predict the mixed dataset indicating that T cannot be calculated using WKB. On the other hand, WKBnet predicts $\ln T$ with less than 1% relative error for the WKB-valid data. Here, WKBnet has estimated a function $F : \mathbb{R}^{2N_V} \to \mathbb{R}$, that can map energy and potential to the log of the transmission probability. In fig. 41 model predictions are shown in a $\ln T$-E graph for an arbitrary test potential with 8 rectangular barriers.

Next, we examine the output of the each FCNN-cell, a_i. Based on our theoretical knowledge about WKB we would expect the FCNN-cell to estimate the following function: $a_i(V_i, E) = -2|k_i|\lambda = -2C\lambda\sqrt{V_i - E}$. Thus, if the network is utilizing WKB to solve the system, a_i should be proportional to the square root of potential and energy difference. In fig. 42a a_i is plotted as a function of the V_i for three different energies, $E = 2$ (blue), $E = 3$ (green) and $E = 4$ (red). We notice that a_i is clearly shifted based on the energy input. Here we see that the curves all nicely follow the graph of a square root. The output dependency further illustrated in fig. 42b, where a_i is plotted against $|k_i|$. We see a linear dependency on $|k_i|$ for the different energy inputs.



(a) Plot of FCNN-cell output a_i vs. V_i input for three different energy inputs.



(b) Plot of FCNN-cell output a_i vs. $|k_i| = 2\sqrt{V_i - E}$ input for three different energy inputs.

Figure 42: Two plots showcasing the information stored in output from the FCNN-cells of WKBnet

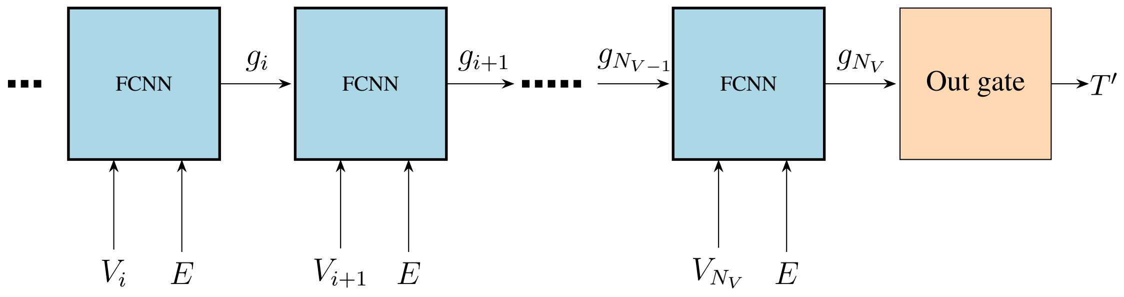


Figure 43: Tranfer-RNN: RNN implemented to predict the probability for a particle passing through a series of rectangular potential barriers.

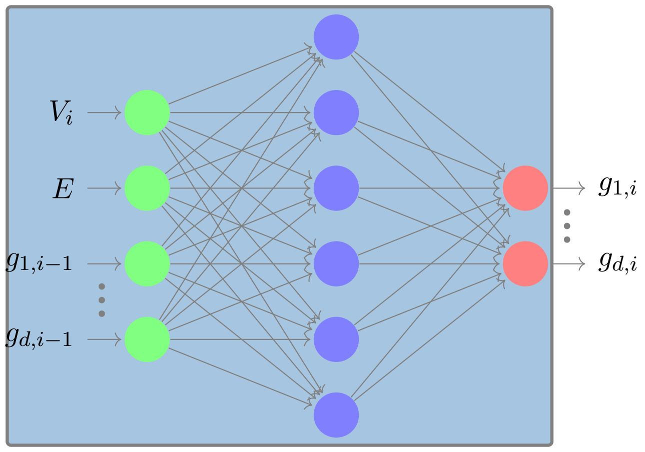


Figure 44: The FCNN-based cell of the Transfer-RNN. At each step in the sequence the FCNN-cell is given an input vector consisting of E, V_i and g_{i-1}, which it maps to the updated hidden state g_i.

Creating a suitable dataset turns out to be the main challenge of successfully training WKBnet. Combinations of V and E, where WKB is valid, limits the diversity of the training dataset. If the majority of $\ln T$ values end up with with an E dependency similar to fig. 41a, the model becomes susceptible to learning a trivial and uninteresting mapping where a good prediction accuracy can be still be reached. We avoid this issue by exposing the model to potentials with a different number of barriers. It increases the variety of $\ln T$, while a smaller interval of V_i values can be used.

Conclusion

WKBnet learns to utilize WKB to accurately predict tunneling probabilities, when trained on a suitable dataset. However, the possible applications of models similar to WKBnet are limited. In reality, we design a specific insufficient model and when the model is trained on the (right) incomplete dataset, information corresponding to the WKB method is stored in the cell output. Obtaining these result requires a great deal of knowledge about both the underlying physical system and meticulous creation of the training data. Hence, it is unlikely that similar sparse models can learn anything useful when exposed to an unknown dataset.

5.3 Transfer-RNN

We now consider the system with no restrictions on the barrier height or particle energy. The provided data is different compared to the previous sections 4 and 3. Each sample now consists of a sequence of inputs but only single scalar output. As a result, the network has to perform a sequence-to-scalar mapping to predict T. A sequence-to-sequence learning architecture can, therefore, no longer be used. We design a new recurrent model, called Transfer-RNN, tailored to the new system. The Transfer-RNN cell only consist of an update law which, at each step, uses V_i and E as input to modify the d-dimensional hidden vector g_{i-1} from the previous step. After being updated at the last barrier (step N_V), the hidden state g_{N_v} is passed to an output gate that estimates the transmission probability, T. The Transfer-RNN structure is depicted in fig. 43. The update law and output gate are both modelled by FCNNs with 1 hidden layer, given by the equations:

Transfer-RNN:

FCNN-based update law:

$$g_i = W^{(U)} * \text{ELU}\left(W^{(H_U)} * [V_i, E, g_{i-1}] + b^{(H_U)}\right) + b^{(U)} \tag{98}$$

Output gate:

$$T' = W^{(T)} * \text{ELU}\left(W^{(H_T)} * g_{N_V} + b^{(H_T)}\right) + b^{(T)}. \tag{99}$$

The FCNN-based update law is given the concatenated vector $[V_i, E, g_{i-1}] \in \mathbb{R}^{d+2}$ as input, which it then maps to the updated hidden state $g_i \in \mathbb{R}^d$. The process is illustrated in fig. 44. The FCNN-based output gate only takes the final hidden state as input, but is otherwise unchanged from the sequence-to-sequence implementation. The initial hidden state g_0 is chosen as an all-ones vector. But we find that arbitrary initial values yields similar results. The model is evaluated based on its ability to predict T using the MSE loss:

$$\mathcal{L}^T_{MSE} = \sum_{i \in \text{batch}} (T' - T)^2 \tag{100}$$

We build a Transfer-RNN model with 100 hidden nodes in the FCNN update law and 200 hidden nodes in the FCNN output gate. The model is trained for different values of d. To learn the most about the underlying system, we search for lowest hidden state dimension where the loss is not sacrificed. Using the Adam optimizer with default learning rate and batch size 50, the different d models are trained for 50000 epochs on the mixed (classical and non-classical) dataset. The training history is shown in fig. 45. We find that the loss is greatly reduced for $d \geq 3$, indicating that storing 3 variables in the hidden state enables the model to accurately estimate the transmission probability. This result is rather surprising since we expected the model to emulate the transfer-matrix approach. In the transfer-matrix approach, the transmitted coefficient (F) is gradually transformed into the reflected and incident wave coefficients (A and B), by the propagation p and discontinuity b matrices. If the update law approximates the b and d matrices, we would expect the hidden states to store the in-between coefficients A_i and B_i. Additionally, the hidden state would have to record the amplitude of the barrier V_i since it is necessary to construct the discontinuity matrix.
$g_{i-1} = (A_{i-1}, B_{i-1}, V_{i-1})^T$ indeed correspond to a 3-dimensional hidden state, however, we consider both regions of $E < V$ and $E > V$. This means that the intermediate coefficients can take complex values, $A_i \in \mathbb{C}, B_i \in \mathbb{C}$. The network is limited to real numbers and would, therefore, require 5 real degrees of freedom to capture the transfer-matrix approach without sacrificing prediction accuracy. However, this can be explained by considering the underlying equation i.e. the time-independent Schrödinger equation (SE). The time-independent SE is entirely real, which means that all stationary state solutions without loss of generality can be chosen as real. Thus, we imagine that when the model is only provided 3 degrees of freedom, it identifies the real solution.

Before studying the hidden variables further, the quality of the $d = 3$ model is examined. We ask the model to predict T for test potentials from the 3 different types of transmission (classical, tunneling and mixed) all for different values of N_V than it was trained on. Fig. 46 depicts T-E plots comparing the prediction and target values for 3 drastically different potential sequences. The first two samples show accurate model predictions on the mixed region data and in fig. 46e predictions for a 'Classical' data sample is plotted. Here, we see that the model even considers interference effects. The validation data predictions yield a mean relative error of:

$$\langle \Delta \rangle_{\text{Transfer}} = 2.25\%$$

Keeping predictions within 5% of the target data (compared to WKBnet with a mean relative error of 164%) , demonstrates the model's versatile prediction power, which generalizes to samples substantially different from

the training data. The first quartile of transmission probabilities (here, corresponding to $T < 0.1$) are not considered in mean relative error calculation. However, as seen in 46c, the model does not show any signs of poor performance for target values close to 0. The model performance strongly implies that the model has captured the governing principles of the system.
The great performance of the $d = 3$ Transfer-RNN indicates that 3 key features stored in the hidden state vector $g_i = (g_{i,1}, g_{i,2}, g_{i,3})^T$ is sufficient to capture transfer-matrix system. We analyse the individual hidden state variables, in an attempt to learn more. First, we look for a possible correlation between g_i and V_i. We do this by plotting the individual g_i components as a function of the potential input for the entire validation set. The plots of the 3 different components are shown in fig. 47. The first-order correlation function between V_i and the components

$$\langle V_i, g_{i,1}\rangle = -0.04 \quad , \quad \langle V_i, g_{i,2}\rangle = -0.47 \quad , \quad \langle V_i, g_{i,3}\rangle = -0.42$$

tells us that V_i and $g_{i,1}$ are (to first-order) uncorrelated, whereas $g_{i,2}$ and $g_{i,2}$ are partially correlated. Thus, the model does seem to store information about V_i in the second and third hidden nodes. However, based on the plot we are unable to interpret an explicit V_i dependency. The reason for this will be addressed in the discussion.
In fig. 48 the hidden state components $g_{i,1}$, $g_{i,2}$ and $g_{i,3}$ are plotted as a function of the sequence step i for a single potential with 35 barriers and 3 different particle energies. The different energies $E = 2$, $E = 10$ and $E = 17$ correspond to tunneling, 'classical' and mixed scattering, respectively. We observe that the evolution of the hidden states, throughout the sequence, are different for the 3 types of transmission. Notably, $g_{i,1}$ shows exponential decay for the $E = 2$, oscillating behaviour for $E = 17$ and a mix between the two for $E = 10$. This corresponds to the evolution of the incident wave amplitude, A, as it passes through the barrier. If A (and B) is transformed into the transmitted amplitude, F, that means the network solves the system in the opposite order of how we generated the data.
This sounds plausible since the two approaches are (in theory) equivalent. It simply corresponds to applying the inverse transfer matrix t^{-1} on both sides in Eq.(90) yielding:

$$\begin{pmatrix} 0 \\ F \end{pmatrix} = t^{-1}\varphi_{\text{left}} = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}^{-1} \begin{pmatrix} B \\ A \end{pmatrix}. \tag{101}$$

This approach would require $g_{i,2}$ and $g_{i,3}$ to jointly store a combination of transformed reflected amplitude B_i and the potential V_i. Based on the values in fig. 48, we cannot determine, if that is the case. The network has the freedom to store any non-linear combination of V_i and B_i in the nodes, which leaves us unable to extract the information.

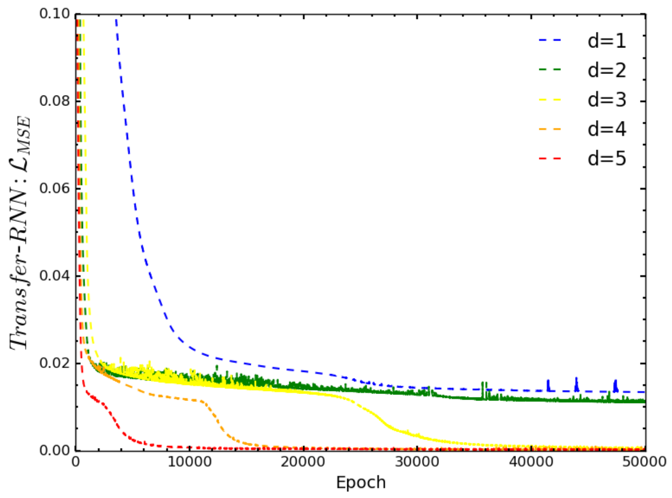
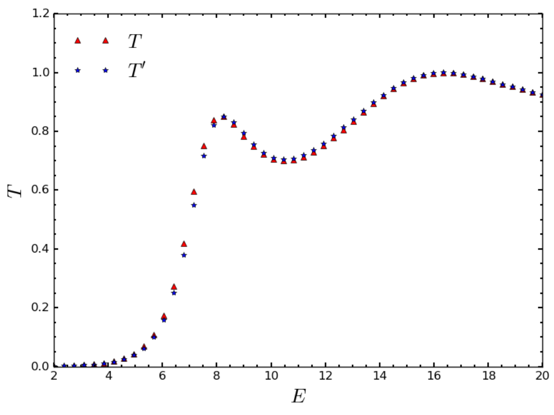
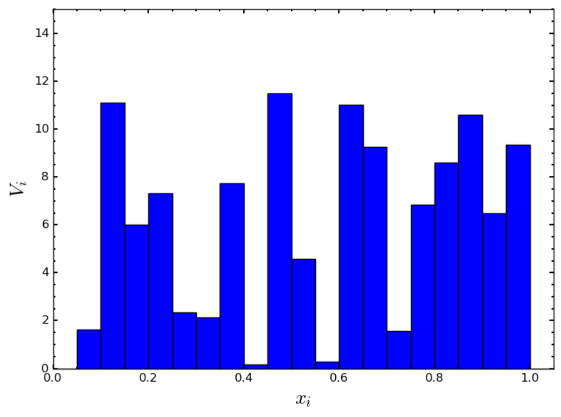


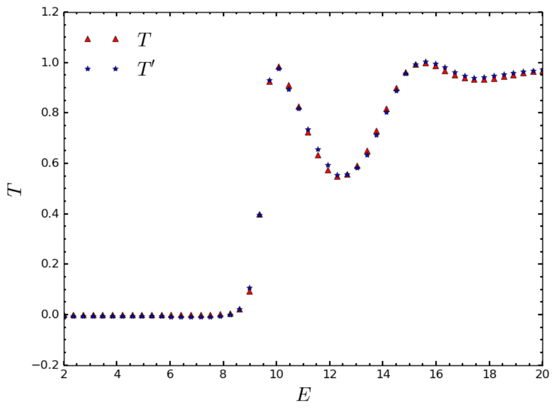
Figure 45: Training history for Transfer-RNN models with different dimension, d, of the hidden state g_i.



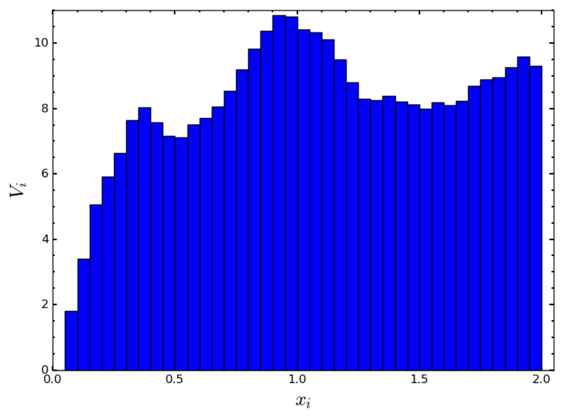
(a) Comparison of target and $d = 3$ model prediction vs E for a 21-barrier test potential shown in b).



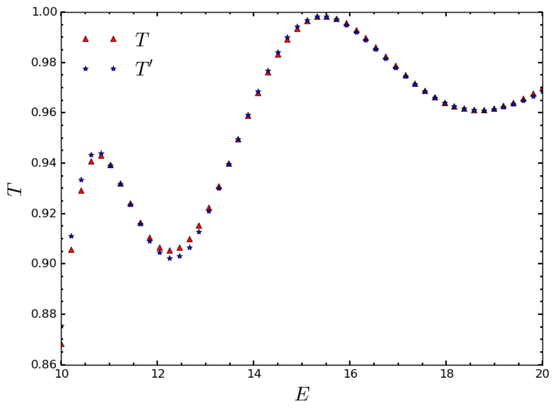
(b) Test potential sequence with 21 barriers, used as input sequence in a).



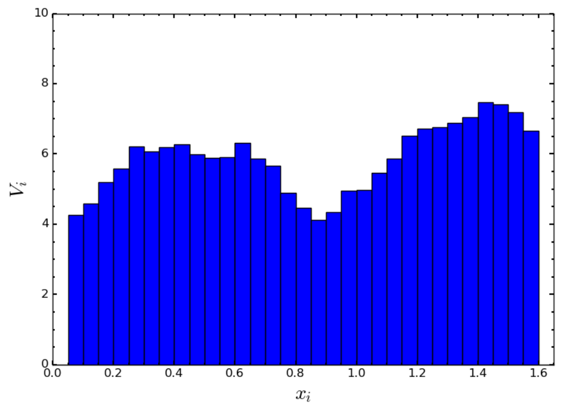
(c) Comparison of target and $d = 3$ Transfer-RNN prediction vs E for a 41-barrier test potential, shown in d).



(d) Test potential sequence with 41 barriers, used as input sequence for the predictions in c).



(e) Transfer-RNN predictions plotted against different energies for a 25-barrier potential shown in d). Sample is from the classical region dataset hence, $E > V\ \forall V_i$.



(f) Test potential sequence with 25 barriers used to make the prediction in e).

Figure 46: Transfer-RNN : 3 different examples of the $d = 3$ Transfer-RNN predictions of transmission probability compared to the target value. The model is able to accurately predict T for different types of input data, not used in the training process.

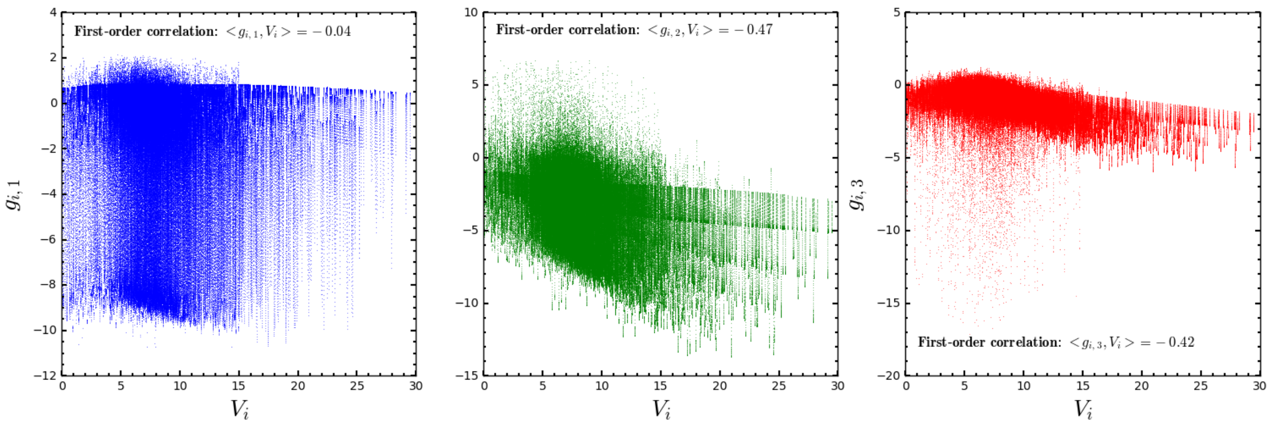


Figure 47: The values stored in the hidden states of the $d = 3$ Transfer-RNN plotted as a function of V_i. This is done to check, if the Transfer-RNN record some variation of V_i in its hidden state.

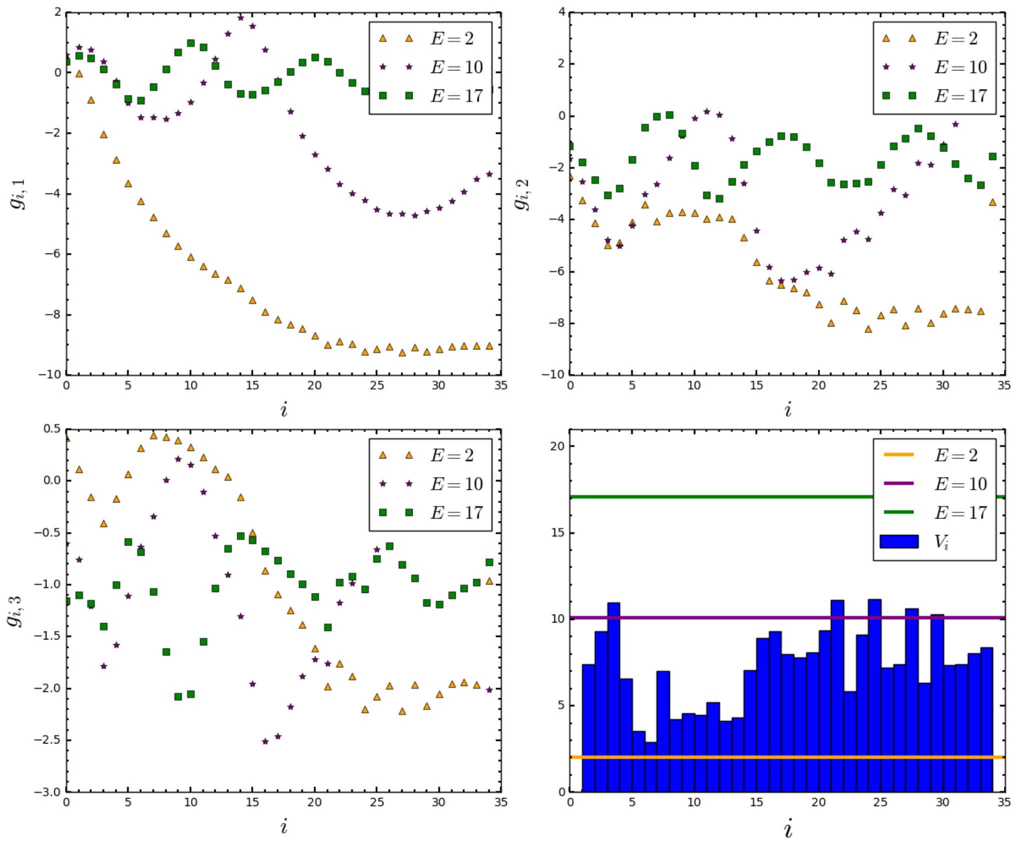


Figure 48: The individual elements of the hidden state $g_i = (g_{i,1}, g_{i,2}, g_{i,3})^T$ for the $d = 3$ Transfer-RNN are plotted at each sequence step i for a single potential sequence ($N_V = 35$). Each hidden node is plotted for 3 different energies indicated by the color and symbol. The last plot shows the potential sequence with the respective particle energies plotted as horizontal lines.

The inverse of the transfer matrix is calculated from the the reverse order of the barrier sequence. Therefore, the evolution $g_{i,1}$ is fascinating because the machine is provided with the same order of the potential sequence as used to generate the data. From a scattering theory, one can show that the transmission probability is the same regardless if the particle is incident from the right or the left. Thus, it is possible that machine determined the transmission probability 'assuming' the particle was incident from the left.
Finally, the transmission probability is determined solely from the transfer-matrix, which means that we do not have the intermediate amplitudes and the left and right moving wave. As a result, the interpretation of the hidden variable become qualitative rather than quantitative.

Conclusion

We have constructed a recurrent network called Transfer-RNN that can predict transmission probabilities of a particle incident on a potential with less than 3% relative error. Furthermore, the network identifies that 3 essential variables need to be stored in the hidden state, to solve the system. However, we are unable to fully extract the relevant physical features of the system.

5.4 Discussion

The Transfer-RNN can predict the transmission probabilities with great accuracy by approximating the discontinuity and propagation matrices (or something equivalent) in the update law. The sequential transformation is highly non-linear in the input, and hence only possible due to the implementation of the FCNN-based update law. However, the versatility of the FCNN is exactly what renders us unable to extract the meaningful features of the system. The issue is the free choice of basis used to express g_i. The Taylor update law subjects the hidden states to a linear transformation, which means all possible bases was related by a linear transformation. However, the FCNN can essentially approximate any (smooth) mapping. This means that the key features of the system can be expressed in an almost arbitrary manner, making the extraction process exceptionally hard.

Current state of the art research on this topic tries to develop methods that force the network to express latent variables in a simple manner. One method that has shown promising results is the use of constrained variational auto encoders (β-VAE) introduced by Google DeepMind researchers in [28]. Variatonal auto encoders are generative versions of the auto encoders introduced in section 3.4. Instead of compressing the input to a single estimate of encoded variables, they encode conditional distributions of the variables. As a result, when the encoder has been trained, encoded variables can be drawn from the learned distribution without needing to provide the encoder with new input. While convenient, VAEs still suffer from the fact that the encoded variables/distributions can be expressed in an arbitrary manner. β-VAEs, on the other hand, have an additional term in their cost function. The added term encourages the hidden variable distributions to be statistically independent and punishes large hidden values. R. Iten et al. have in [29] developed a neural network architecture called *sci-net*. Which by utilizing a β-VAE, successfully manages to extract the relevant features from simple physical systems.

The approach is not directly applicable to the Transfer-RNN, where no auto-encoder is utilized. But encouraging the network to express hidden states in a simple manner through regularization is a promising concept. We attempted to implement this for the Transfer-RNN by adding a term that punished large hidden state values. Within the time frame of this thesis, we did not manage to train a model with accurate predictions based on this approach. Recurrent models are, due to vanishing/exploding gradients, difficult to train. Adding a regularization term increases the difficulty even further. However, neural learning is largely based on trial and error. Therefore, it is possible that a balance between regularization and prediction error exits, which can train a model to accurate predictions. Additionally, in future works, it could be interesting to construct a learning architecture inspired by the one from section 3 and 4, where the prediction task and regularization is separated.

The complete solution method cannot be extracted from a FCNN-based update law but we might be able to extract the important physical features, which in itself could be considered important discoveries.

6 Conclusion and Outlook

In this thesis, we have explored the prospects of using neural networks to discover physical concepts from data where the governing laws of the underlying system might be unknown. We did this, by considering simulated data of three simple quantum mechanical systems.

As a starting point, we considered analytical extended state solutions to the Schrödinger equation for training a neural network. This was first investigated by Zhai et al. in [1]. They developed a recurrent learning architecture, called the *Schrödinger Machine*, which consisted of two components. The first component was a recurrent neural network, capable of performing a potential-to-density mapping. The second component was a recurrent autoencoder, which stored the wavefunction and its derivative in the hidden states. The function, updating the hidden states, turned out to be the machine's formulation of the discrete Schrödinger equation. We have managed to reproduce the results of Zhai et al. and additionally discovered that this architecture is only able to solve the system when the constant slabs of potential are sufficiently narrow relative to their height. Under that condition, a numerical solution is equivalent to the analytical solution from the dataset.

We then proceeded to construct an extended version of the learning architecture that incorporated a global encoder, designed to identify essential non-local variables, which the recurrent architecture could not discover. This allowed the learning architecture to predict the probability distribution of ground state solutions to the discrete Schrödinger equation with infinite potential boundary conditions. Here the global encoder was able to pinpoint the ground state energy as the essential global variable needed to perform the potential-to-sequence mapping. By providing the recurrent autoencoder with the global variable, it recognized the wavefunction and its difference quotient as the necessary hidden variables, while developing the discrete Schrödinger equation as the update law.

The recurrent learning architecture developed has one key quality. The method it utilizes to make predictions is fully transparent. We have shown, however, that conserving this property means that the architecture is unsuitable for systems where complicated non-linear update laws are required. Based on this, we will have to reconsider our understanding of what scientific discoveries entail, in regards to the recurrent neural network approach. The recurrent learning architecture managed to capture the discrete Schrödinger equation. However, the machine was in some ways set up for success. The only reason we could verify that the update law of the network corresponded to the discrete Schrödinger equation was due to the update law having a linear dependency on the potential.

Many neural networks have an inherent bias in the types of solutions they can produce, based on the way they are designed, which logically limits their ability to discover concepts foreign to the human imagination. Learning Machines do not possess any preconceived understanding of the world and if applied properly they can potentially help formulate physics in terms not thought of before. We have to invent ways to utilize that quality and not confine discoveries to the human way of thinking. Thus, insisting on keeping track of the complete solution method used by the network is not necessarily the right approach.

In section 5, the effects of using a non-linear update equation were showcased by the Transfer-RNN, which was designed with a FCNN-based update law. The model learned to predict the transmission probabilities of a particle scattering of a potential, when provided the potential sequence and particle energy as input. We discovered that 3 variables needed to be stored in the hidden states of the Transfer-RNN to make accurate predictions. However, we were unable to extract precisely what physical features these variables corresponded to. A regularization term was introduced to the cost function to encourage a more interpretable formulation of the hidden variables. We did not manage to successfully train a Transfer-RNN in this way, however.

In future works, it would be interesting to construct a learning architecture, where the regularization of the hidden variables is separated from the prediction task of the model. We envision this would allow the regularization component to discover an interpretable formulation of the hidden states without compromising the prediction loss. While the principle is similar to the Schrödinger Machine, this would not be an autoencoder. Both components should be optimized with the minimal size of the hidden state needed for an accurate prediction.

References

[1] H. Zhai C. Wang and Y. You. Emergent Schrödinger equation in an introspective machine learning architecture. *Science Bulletin*, 64(20):1228–1233, 2019.

[2] Y. LeCun et al. Deep learning. *Nature*, 521:436–444, 2015.

[3] G. Neubig. Neural Machine Translation and Sequence-to-sequence Models: A Tutorial. *arXiv preprint*, arXiv:1703.01619, 2017.

[4] J. Carrasquilla and R. Melko. Machine Learning Phases of Matter. *Nature Phys*, 13:431–434, 2017.

[5] E. van Nieuwenburg et al. Learning phase transitions by confusion. *Nature Phys*, 13:435–439, 2017.

[6] G. Carleo and M. Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355:602–606, 2017.

[7] N. Yusuke et al. Restricted boltzmann machine learning for solving strongly correlated quantum systems. *Phys. Rev. B*, 96:205152, 2017.

[8] G.Torlai et al. Neural-network quantum state tomography. *Nature Phys*, 14:447–450, 2018.

[9] M.A. Nielsen. *Neural Networks and Deep Learning* . Determination Press, 2015.

[10] I. Goodfellow et al. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] D. Clevert et al. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[12] L. Zhou et al. The Expressive Power of Neural Networks: A View from the Width. *Advances in Neural Information Processing Systems*, 30, 2017.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems*, 2:303–314, 1989.

[14] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint*, 2017.

[15] J. Duchi et al. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, 2011.

[16] G. Hinton. Neural Networks for Machine Learning - Lecture 6a.

[17] J. Martens. Deep learning via Hessian-free optimization. In L. Bottou and M. Littman, editors,. *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

[18] D. J. Giffiths. *Introduction to Quantum Mechanics*. Cambridge University Press, 2017.

[19] S. Hochreiter and J. Schimdhuber. Long Short Term Memory. *Science Bulletin*, 9(8):1735–1780, 1997.

[20] K. Cho et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014.

[21] Y. Bengio et al. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[22] J. J. Sakurai. *Modern quantum mechanics; rev. ed.* Addison-Wesley, 1994.

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *PMLR*, 37:448–456, 2015.

[24] S. Kirkpatrick et al. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[25] R. Barlow. *Statistics: a guide to the use of statistical methods in the physical sciences.* John Wiley and Sons, 1989.

[26] A.Iserles. *First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 1996.

[27] J.S. Walker and J. Gathright. A transfer-matrix approach to one-dimensional quantum mechanics using Mathematica. *Computers in Physics*, 6(393), 1992.

[28] Irina Higgins et al. Beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2017.

[29] R. Iten et al. Discovering physical concepts with neural networks. *Phys. Rev. Lett. 124, 010508*, 2020.

Appendix

A Appendix

A.1 Derivation of Relations for Constants A and B

We will here derive the recurrence relations for A and B in chapter.
Matching the wave function $\psi(\mathrm{x})$ at the $\mathrm{x}_{i+1} = a(i+1)$ boundary, we get:

$$\psi(\mathrm{x}_{i+1}) = A_i \sin(k_i \mathrm{x}_{i+1}) + B_i \cos(k_i \mathrm{x}_{i+1}) = A_{i+1} \sin(k_{i+1} \mathrm{x}_{i+1}) + B_{i+1} \cos(k_{i+1} \mathrm{x}_{i+1}) \tag{102}$$

and matching the derivatives at x_{i+1} gives:

$$\partial\psi(\mathrm{x})|_{\mathrm{x}=\mathrm{x}_{i+1}} = k_i A_i \cos(k_i \mathrm{x}_{i+1}) - k_i B_i \sin(k_i \mathrm{x}_{i+1}) = k_{i+1} A_{i+1} \cos(k_{i+1} \mathrm{x}_{i+1}) - k_{i+1} B_{i+1} \sin(k_{i+1} \mathrm{x}_{i+1}) \tag{103}$$

Isolating B_{i+1} in (102)

$$B_{i+1} = \frac{1}{\cos(k_{i+1}\mathrm{x}_{i+1})}[A_i \sin(k_i \mathrm{x}_i) + B_i \cos(k_i \mathrm{x}_{i+1}) - A_{i+1} \sin(k_{i+1} \mathrm{x}_{i+1})] \tag{104}$$

Similarly, we isolate A_{i+1} in (103)

$$A_{i+1}k_{i+1} = \frac{1}{\cos(k_{i+1}\mathrm{x}_{i+1})}[k_i(A_i \cos(k_i \mathrm{x}_{i+1}) - B_i \sin(k_i \mathrm{x}_{i+1})) + B_{i+1}k_{i+1} \sin(k_{i+1} \mathrm{x}_{i+1})] \tag{105}$$

Now inserting eq.(104) into eq.(105)

$$\begin{aligned} A_{i+1}k_{i+1} = \frac{1}{\cos(k_{i+1}\mathrm{x}_{i+1})} \Bigg[& k_i(A_i \cos(k_i \mathrm{x}_{i+1}) - B_i \sin(k_i \mathrm{x}_{i+1})) + \frac{k_{i+1} \sin(k_{i+1}\mathrm{x}_{i+1})}{\cos(k_{i+1}\mathrm{x}_{i+1})} \\ & [A_i \sin(k_i \mathrm{x}_{i+1}) + B_i \cos(k_i \mathrm{x}_{i+1}) - A_{i+1} \sin(k_{i+1} \mathrm{x}_{i+1})] \Bigg] \end{aligned} \tag{106}$$

Moving $A_{i+1}k_{i+1}$ terms to the LHS

$$\begin{aligned} A_{i+1}k_{i+1}\left(1 + \frac{\sin^2(k_{i+1}\mathrm{x}_{i+1})}{\cos^2(k_{i+1}\mathrm{x}_{i+1})}\right) = \frac{1}{\cos(k_{i+1}\mathrm{x}_{i+1})} \Bigg[& k_i(A_i \cos(k_i \mathrm{x}_{i+1}) - B_i \sin(k_i \mathrm{x}_{i+1})) \\ & + \frac{k_{i+1} \sin(k_{i+1}\mathrm{x}_{i+1})}{\cos(k_{i+1}\mathrm{x}_{i+1})}[A_i \sin(k_i \mathrm{x}_{i+1}) + B_i \cos(k_i \mathrm{x}_{i+1})] \Bigg] \end{aligned} \tag{107}$$

Finally we can utilize the trigonometric identity $\cos^2(x) + \sin^2(x) = 1$ such that

$$1 + \frac{\sin^2(k_{i+1}\mathrm{x}_{i+1})}{\cos^2(k_{i+1}\mathrm{x}_{i+1})} = \frac{\cos^2(k_{i+1}\mathrm{x}_{i+1}) + \sin^2(k_{i+1}\mathrm{x}_{i+1})}{\cos^2(k_{i+1}\mathrm{x}_{i+1})} = \frac{1}{\cos^2(k_{i+1}\mathrm{x}_{i+1})}$$

Hence we (107) becomes

$$\begin{aligned} \frac{A_{i+1}k_{i+1}}{\cos^2(k_{i+1}\mathrm{x}_{i+1})} = \frac{1}{\cos(k_{i+1}\mathrm{x}_{i+1})} \Bigg[& k_i(A_i \cos(k_i \mathrm{x}_{i+1}) - B_i \sin(k_i \mathrm{x}_{i+1})) \\ & + \frac{k_{i+1} \sin(k_{i+1}\mathrm{x}_{i+1})}{\cos(k_{i+1}\mathrm{x}_{i+1})}[A_i \sin(k_i \mathrm{x}_{i+1}) + B_i \cos(k_i \mathrm{x}_{i+1})] \Bigg] \end{aligned} \tag{108}$$

$\Updownarrow$

$$\boxed{\begin{aligned}A_{i+1}k_{i+1} = k_i\Big(&A_i\cos(k_i\mathrm{x}_{i+1})\cos(k_{i+1}\mathrm{x}_{i+1}) - B_i\sin(k_i\mathrm{x}_{i+1})\cos(k_{i+1}\mathrm{x}_{i+1})\Big)\\ +k_{i+1}\Big(&A_i\sin(k_i\mathrm{x}_{i+1})\sin(k_{i+1}\mathrm{x}_{i+1}) + B_i\cos(k_i\mathrm{x}_{i+1})\sin(k_{i+1}\mathrm{x}_{i+1})\Big)\end{aligned}} \tag{109}$$

The derivation is almost identical for $B_{i+1}k_{i+1}$, done by isolating $B_{i+1}k_{i+1}$ in eq. (103) now instead.

$$B_{i+1}k_{i+1} = \frac{1}{\sin(k_{i+1}\mathrm{x}_{i+1})}[k_i(B_i\sin(k_i\mathrm{x}_{i+1}) - A_i\cos(k_i\mathrm{x}_{i+1})) + A_{i+1}k_{i+1}\cos(k_{i+1}\mathrm{x}_{i+1})] \tag{110}$$

Inserting eq.(109) we arrive at

$$\begin{aligned}B_{i+1}k_{i+1} = &\frac{k_i(B_i\sin(k_i\mathrm{x}_{i+1}) - A_i\cos(k_i\mathrm{x}_{i+1}) + \cos^2(k_{i+1}\mathrm{x}_{i+1})(A_i\cos(k_i\mathrm{x}_{i+1}) - B_i\sin(k_i\mathrm{x}_{i+1})))}{\sin(k_{i+1}\mathrm{x}_{i+1})}\\ &+\frac{k_{i+1}\cos(k_{i+1}\mathrm{x}_{i+1})}{\sin(k_{i+1}\mathrm{x}_{i+1})}\Big(A_i\sin(k_i\mathrm{x}_{i+1})\sin(k_{i+1}\mathrm{x}_{i+1}) + B_i\cos(k_i\mathrm{x}_{i+1})\sin(k_{i+1}\mathrm{x}_{i+1})\Big)\end{aligned} \tag{111}$$

$\Updownarrow$

$$\begin{aligned}B_{i+1}k_{i+1} = &\frac{k_i(B_i\sin(k_i\mathrm{x}_{i+1}) - A_i\cos(k_i\mathrm{x}_{i+1}))(1-\cos^2(k_{i+1}\mathrm{x}_{i+1}))}{\sin(k_{i+1}\mathrm{x}_{i+1})}\\ &+k_{i+1}\Big(A_i\sin(k_i\mathrm{x}_{i+1})\cos(k_{i+1}\mathrm{x}_{i+1}) + B_i\cos(k_i\mathrm{x}_{i+1})\cos(k_{i+1}\mathrm{x}_{i+1})\Big)\end{aligned} \tag{112}$$

Using the same identity $\sin^2(x) = 1 + \cos^2(x)$ we end up with the final expression.

$$\boxed{\begin{aligned}B_{i+1}k_{i+1} = k_i\Big(&B_i\sin(k_i\mathrm{x}_{i+1})\sin(k_{i+1}\mathrm{x}_{i+1}) - A_i\cos(k_i\mathrm{x}_{i+1})\sin(k_{i+1}\mathrm{x}_{i+1})\Big)\\ +k_{i+1}\Big(&B_i\cos(k_i\mathrm{x}_{i+1})\cos(k_{i+1}\mathrm{x}_{i+1}) + A_i\sin(k_i\mathrm{x}_{i+1})\cos(k_{i+1}\mathrm{x}_{i+1})\Big)\end{aligned}} \tag{113}$$

A.2 Code for Data Generation of Simplified Schrödinger System

```python
import numpy as np, import pandas as pd, import matplotlib.pyplot as plt

## Initial conditions
r=np.random
A_0 = 1
B_0 = 1
V_0 = -1
a = 0.1 ## potential interval
N_in = 400
N_data = 15000

def mk_V_data(V_0,N_in,N_data=None):
    if N_data==None:
        N_data = 500
    V = np.zeros((N_data,N_in))
    for i in range(N_data):
        R = r.rand(1)
        V[i,:] = -2*r.rand(N_in) - R # R shifts the energy scale
    V[:,0] = V_0

    return V

def smoothing_V(q,V):
    for p in range(len(V[:,0])):
        for j in range(r.randint(q)): #use r.randint(q) for random NoT
         ↪  smoothing for each sample
            for i in range(len(V[0,:])-1):
                V[p,i+1] = 0.5*(V[p,i]+V[p,i+1])
    return V

# making potential
V = mk_V_data(V_0,N_in,N_data)

q=20 # smoothing the potential q times
V = smoothing_V(q,V)#smoothing potential

#### finding all A's and B's
A,B = np.zeros((N_data,N_in)),np.zeros((N_data,N_in)) ## zero array for all
 ↪  needed coefficients
A[:,0],B[:,0] = A_0,B_0 # fixed first coeff
for i in range(N_in-1): ## for loop to find each A and B one at a time

    A[:,i+1] = A[:,i] * (k[:,i+1] * np.sin(k[:,i] * (i+1) * a)
     ↪  *np.sin(k[:,i+1] * (i+1) * a) + k[:,i] * np.cos(k[:,i] * (i+1) * a) *
     ↪  np.cos(k[:,i+1] * (i+1) * a)) + B[:,i] * (k[:,i+1] * np.cos(k[:,i] *
     ↪  (i+1) * a) * np.sin(k[:,i+1] * (i+1) * a) - k[:,i] * np.sin(k[:,i] *
     ↪  (i+1) * a) * np.cos(k[:,i+1] * (i+1) * a))

    B[:,i+1] = B[:,i] * (k[:,i] * np.sin(k[:,i] * (i+1) * a) *
     ↪  np.sin(k[:,i+1] * (i+1) * a ) + k[:,i+1] * np.cos(k[:,i] * (i+1) * a)
     ↪  * np.cos(k[:,i+1] * (i+1) * a)) + A[:,i] * (k[:,i+1] * np.sin(k[:,i]
     ↪  * (i+1) * a) * np.cos(k[:,i+1] * (i+1) * a) - k[:,i] * np.cos(k[:,i]
     ↪  * (i+1) * a) * np.sin(k[:,i+1] * (i+1) * a))

    A[:,i+1] = A[:,i+1]/k[:,i+1]
    B[:,i+1] = B[:,i+1]/k[:,i+1]

psi = A*np.sin(k*aa)+B*np.cos(k*aa)## finding psi
rho = psi**2 ## square of psi is the density
```

A.3 Basis and Similarity Transformations

Variables stored in the hidden state of the recurrent network models can be represented by a class of bases, i.e. any linear (also arbitrarily scaled) combination of the variables are valid. We need to transform the states to the standard basis to evaluate the models. In order to do that certain principles of linear algebra can be utilized. Let $A \in \mathbb{R}^d$ be the basis of the current basis, and $B \in \mathbb{R}^d$ be the basis of we wish to change coordinates to The change of basis matrix is then given by:

$$M = B^{-1}A$$

Thus when given a vector in the A basis $\mathrm{x_A}$ the matrix operation

$$\mathrm{x_B} = M\mathrm{x_A}$$

transform the vector to the new basis. The linear transformation of the update laws will also naturally be affected by this basis transformation. In linear algebra, matrices can represent the same linear map under two different bases. Two matrices W and $\tilde{W}$ are called similar if there exists an invertible matrix M such that

$$\widetilde{W} = M^{-1}WM$$

with M being the change of basis matrix. A transformation $W \to M^{-1}WM$ is called a similarity transformation [22]. We can utilize this to bring the hidden state and update law on a form where we can make the connection to the standard form of the Schrödinger equation. However, we do not know the basis A that the trained networks has used to represent the values. This issue is handled by manually constructing the change of basis matrix M that bring the a weight matrices $\tilde{W}^{(n)}$ on the wanted form. Afterwards hidden states h_i are examined if all vectors and matrices can be brought to the wanted form (standard basis) with the same M.

A.4 Derivation of Schrödinger Equation in Terms of Probability

We want to derive the three 1. order differential equations leading to Eq.(48). The first differential equation is found trivially by

$$\partial_x\rho(x) = \psi(x)^*\partial_x\psi(x) + (\psi(x)^*\partial_x\psi(x))^* = 2\,\mathrm{Re}(\psi(x)^*\partial_x\psi(x))$$

Hence

$$\partial_x\rho(x) = 2\eta(x) \tag{114}$$

The second equation can be derived using Eq.(44) where we recognize $\xi(x) = \partial_x\psi(x)\partial_x\psi(x)^* = |\partial_x\psi(x)|^2$ and use the derivative

$$\partial_x^2\rho(x) = 2\partial_x\eta(x) = 2V(x)\rho(x) + 2|\partial_x\psi(x)|^2 \Rightarrow$$

$$\partial_x\eta(x) = V(x)\rho(x) + \xi(x) \tag{115}$$

For the final equation we look at

$$\partial_x\xi(x) = \partial_x^2\psi(x)\partial\psi(x)^* + \partial_x^2\psi(x)^*\partial\psi(x)$$

We have to show that this is equivalent to $2V(x)\eta(X)$ to do this we see from Eq.(115) that $V(x)$ can be expressed as

$$V(x) = \frac{\partial_x\eta(x) - \xi(x)}{\rho(x)}$$

where

$$\partial_x\eta(x) = \frac{1}{2}\partial_x(\psi(x)\partial_x\psi(x)^* + \psi(x)^*\partial_x\psi(x)) = \partial_x\psi(x)\partial_x\psi(x)^* + \frac{1}{2}(\psi(x)\partial_x^2\psi(x)^* + \psi(x)^*\partial_x^2\psi(x))$$

Hence $2V(x)\eta(x)$ becomes

$$2\frac{\partial_x\eta(x) - \xi(x)}{\rho(x)}\eta(x) = \frac{\frac{1}{2}(\psi(x)\partial_x^2\psi(x)^* + \psi(x)^*\partial_x^2\psi(x))(\psi(x)\partial_x\psi(x)^* + \psi(x)^*\partial_x\psi(x))}{\psi(x)\psi(x)^*}$$

$$= \frac{\frac{1}{2}(|\psi(x)|^2\partial_x\psi(x)\partial_x^2\psi(x)^* + |\psi(x)|^2\partial_x\psi(x)^*\partial_x^2\psi(x) + \psi(x)^2\partial_x\psi(x)^*\partial_x^2\psi(x)^* + (\psi(x)^*)^2\partial_x\psi(x)\partial_x^2\psi(x))}{\psi(x)\psi(x)^*}$$

Unfortunately, there is no way to reduce this to $\partial_x \xi(x)$ as long as $\psi(x) \in \mathbb{C}$. However, when we use $\psi(x) = \psi(x)^*$ we see that the expression becomes

$$2\frac{\psi(x)^2 \partial_x \psi(x) \partial_x^2 \psi(x)}{\psi(x)^2} = 2\partial_x \psi(x) \partial_x^2 \psi(x)$$

which is equal to

$$\partial_x \xi(x) = \partial_x^2 \psi(x) \partial \psi(x)^* + \partial_x^2 \psi(x)^* \partial \psi(x) = 2\partial_x \psi(x) \partial_x^2 \psi(x), \qquad \text{for } \psi(x) \in \mathbb{R}$$

This way giving us the final first order differential equation

$$\partial_x \xi(x) = 2V(x)\eta(x), \qquad \text{for } \psi(x) \in \mathbb{R} \tag{116}$$

But one has to keep in mind that this form of the Schrödinger equation is only valid for systems where ψ can be considered real with no loss of generality.

B Appendix

B.1 Code for Generating Discrete Ground State Solutions

Numerical solution to infinite wall boundary conditions

```
import numpy as np
import pandas as pd
from numpy import linalg as LA
from numpy import fill_diagonal as fill
import  matplotlib.pyplot as plt
import scipy.constants as c



x_start = 0
x_end = 1
N_steps = 500
N_data = 15000
n_rep = 50
x_d = x_end / ( N_steps + 1 ) ###  +1  because first and last point from
↪  boundary conditions
amplitude = 20


r=np.random
def mk_V_data(N_in,Amp,N_data=None):
    if N_data==None:
        N_data = 500
    V = np.zeros((N_data,N_in))
    for i in range(N_data):

        V[i,:] = Amp*r.rand(N_in)

    return V

def smoothing_V(q,V):
    for p in range(len(V[:,0])):
        for j in range(r.randint(q)): ### to mix up smoothing factor use
        ↪  r.randint(q)
            for i in range(len(V[0,:])-1):
                V[p,i+1] = 0.5*(V[p,i]+V[p,i+1])
    return V

### making the potentials
#V = mk_V_data(V_0,N_steps,amplitude,N_data)
V =np.random.rand( N_data, int(N_steps/n_rep) ).repeat(n_rep,axis=1)
↪  *amplitude



#smoothing potential
q=20 # smoothing the potential q times
V = smoothing_V(q,V)


```

```python
## Set up H-matrix

def H_mat(N,delta_x,V_arr):
    filler = np.ones(N-1)*(1/delta_x**2)
    T =  np.eye(N,N)*2*(1/delta_x**2)
    fill(T[1:,:],-filler)
    fill(T[:,1:],-filler)
    V = np.zeros((N,N))
    fill(V,V_arr)

    return T + V



H = np.zeros((N_data,N_steps,N_steps))
for l in range(N_data):
    H[l,:,:] = H_mat(N_steps,x_d,V[l,:])


### diagonalizing H and finding eigenvectors
E,psi = LA.eig(H)


index_E_s= E.argsort() ### sorting the index of based on energy value for
↪  picking out the different quantized states


E_s= np.sort(E) ## sorting energies

Gs_E = E_s[:,0] ## ground state
E1_E = Gs_E = E_s[:,1] #exited state


gs_psi = np.zeros((N_data,N_steps)) ## allocating space for the groundstates
for i in range(N_data):
    gs_psi[i,:] = psi[i,:,index_E_s[i,0]] #### picking out the lowest energy
     ↪  state in each generated potential.




E1s_psi = np.zeros((N_data,N_steps)) ## allocating space
for i in range(N_data):
    E1s_psi[i,:] = psi[i,:,index_E_s[i,1]] #### picking out the 1. excited
     ↪  state in each generated potential.



psiGS = np.sqrt((gs_psi[:,:]*(1/np.sqrt(x_d)))**2) ### normalizing
↪  groundstates

psiE1 = np.sqrt((E1s_psi[:,:]*(1/np.sqrt(x_d)))**2) ### normalizing first
↪  excited state
```

C Appendix

C.1 Data Generation Parameters Scattering System

Training dataset

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	5	0.05	30	0	20	2	50
Classical							
	5	0.05	9.999	0	20	10	50
Tunneling							
	5	0.05	35	10.001	10	5	30

Table 5: Parameters for the Transmission coefficient training data generation with 5 rectangular potential barriers i.e $N_V = 5$.

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	31	0.05	15	0	20	2	50
Classical							
	31	0.05	9.999	0	20	10	50
Tunneling							
	31	0.05	13	10.001	10	5	30

Table 6: Parameters for the Transmission coefficient training data generation with 31 rectangular potential barriers i.e $N_V = 31$.

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	35	0.05	15	0	20	2	50
Classical							
	35	0.05	9.999	0	20	10	50
Tunneling							
	35	0.05	13	10.001	10	5	30

Table 7: Parameters for the transmission coefficient training data generation with 35 rectangular potential barriers i.e $N_V = 35$.

Validation dataset

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	15	0.05	20	0	20	2	50
Classical							
	15	0.05	9.999	0	20	10	50
Tunneling							
	15	0.05	25	10.001	10	5	30

Table 8: Parameters for the transmission coefficient validation data generation with 15 rectangular potential barriers i.e $N_V = 15$.

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	21	0.05	15	0	20	2	50
Classical							
	21	0.05	9.999	0	20	10	50
Tunneling							
	21	0.05	20	10.001	10	5	30

Table 9: Parameters for the transmission coefficient validation data generation with 21 rectangular potential barriers i.e $N_V = 21$.

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	25	0.05	15	0	20	2	50
Classical							
	25	0.05	9.999	0	20	10	50
Tunneling							
	25	0.05	15	10.001	10	5	30

Table 10: Parameters for the transmission coefficient validation data generation with 25 rectangular potential barriers i.e $N_V = 25$.

Transmission	N_V	λ	V_{max}	V_{min}	E_{max}	E_{min}	N_E
Mixed							
	41	0.05	15	0	20	2	50
Classical							
	41	0.05	9.999	0	20	10	50
Tunneling							
	41	0.05	12	10.001	10	5	30

Table 11: Parameters for the transmission coefficient validation data generation with 41 rectangular potential barriers i.e $N_V = 41$.

C.2 Derivation of WKB

Given Eq.(67) on the form

$$\partial_x^2 \psi(x) = -\frac{k^2}{\hbar^2}\psi(x),$$

where $k = \sqrt{2m(E - V(x))}$ We follow [18] for the derivation, motivated by free-particle (constant V) solution we write a solution the SE in terms of its amplitude $A(x) \in \mathbb{R}$ and phase $\phi(x) \in \mathbb{R}$:

$$\psi(x) = A(x)\,\mathrm{e}^{i\phi(x)}\,.$$

Inserting this into the SE we and calculating the first and second order derivatives, we find:

$$\partial_x^2 A(x) + 2i\partial_x A(x)\partial_x\phi(x) + iA(x)\partial_x^2\phi(x) - A(x)\left(\partial_x\phi(x)\right)^2 = -A(x)\frac{k^2}{\hbar^2}.$$

This equation can be split into two equations; one for the real part and one for the imaginary part.
Real part:

$$\partial_x^2 A(x) - A(x)\left(\partial_x\phi(x)\right)^2 = -A(x)\frac{k^2}{\hbar^2}$$

Imaginary part:

$$2i\partial_x A(x)\partial_x\phi(x) + iA(x)\partial_x^2\phi(x) = 0 \Rightarrow \partial_x(A(x)^2\partial_x\phi(x)) = 0$$

The imaginary equation can solved analytically with:

$$A(x) = \frac{C}{\sqrt{\partial_x\phi(x)}},$$

where C is a real (for now) constant. The real equation has no analytical solution. To solve it, we assume that $A(x)$ changes slowly i.e. the second order derivative $\partial_x^2 A(x)$ is sufficiently small that it can be ignored. Thus, the real equation becomes:

$$\partial_x\phi(x) = \pm\frac{k}{\hbar},$$

hence,

$$\phi(x) = \pm\frac{1}{\hbar}\int k(x)\,dx,$$

is a solution. Combining the two and absorbing constants into C (hence $C \in \mathbb{C}$) we find a WKB solution for $\psi(x)$:

$$\psi(x) \simeq \frac{C}{\sqrt{k(x)}}\,\mathrm{e}^{\pm\frac{i}{\hbar}\int k(x)\,dx} \tag{117}$$

We intend to make us of WKB in *non*classical regions where $E < V(x)$. As a result, $k(x)$ becomes imaginary and $\psi(x)$ yields:

$$\psi(x) \simeq \frac{C}{\sqrt{|k(x)|}}\,\mathrm{e}^{\pm\frac{1}{\hbar}\int |k(x)|\,dx} \tag{118}$$

The solution can be used to with determine scattering/transmission from potential barriers. Consider an incident and reflected wave to the left of a potential barrier placed at $x = 0$.

$$\psi(x) = A\,\mathrm{e}^{i\kappa x} + B\,\mathrm{e}^{-i\kappa x}, \quad x < 0,$$

where $\kappa = \sqrt{2mE}/\hbar$. The the transmitted wave to the right of the barrier (ending at $x = L$) is given by

$$\psi(x) = F\,\mathrm{e}^{i\kappa x}, \quad x > L.$$

The relative amplitude between the incident and transmitted wave is given by the exponential decay in the nonclassical region. Thus,

$$\frac{|F|}{|A|} = \mathrm{e}^{-\frac{1}{\hbar}\int_0^L |k(x)|\, dx}.$$

From this, we find that the transmission coefficient, T, becomes:

$$T = \frac{|F|^2}{|A|^2} = \mathrm{e}^{-2\gamma}, \quad \text{where} \quad \gamma = \frac{1}{\hbar}\int_0^L |k(x)|\, dx \tag{119}$$

In this thesis we set $\hbar = 1$ and $m = 1$.