Department of Informatics

Technical University of Munich

Master's Thesis in
Robotics, Cognition, Intelligence

Machine Learning Techniques for Simulating Quantum Dynamics

Amr Ibrahim

Department of Informatics

Technical University of Munich

Master's Thesis in
Robotics, Cognition, Intelligence

Machine Learning Techniques for Simulating Quantum Dynamics

Maschinelles Lernen Methoden zur Simulation von Quantumdynamik

Author:	Amr Ibrahim
Supervisor:	Univ.-Prof. Dr. Christian Mendl
Advisor:	M.Sc. Irene Lopez Gutierrez
Submission Date:	Mar 15th, 2021

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Mar 15th, 2021 *Amr Ibrahim*

Acknowledgments

The code used in this work has been partially based on code written by my supervisors Dr. Mendl and M.Sc. Gutierrez, and I would like to thank them for their support and consideration. I would also like to extend my thanks to Dr. Renjewski and the program coordinator Dr. Lenz for their understanding.

The format of this work has been largely based on a template published by Yoshiyuki Sakai and Walter Simons. I therefore would like to thank them for their contributions.

"Twenty years from now you will be more disappointed by the things you didn't do than by the ones you did do. So, throw off the bowlines. Sail away from the safe harbor. Catch the trade winds in your sails. Explore. Dream. Discover."

-Mark Twain

In loving memory of my sister

Abstract

Simulating quantum dynamics is considered to be intractable on a classical computer and the probabilistic properties of nature make it even harder to predict the outcome of a simulation. This thesis investigates the efficacy of machine learning techniques to solve the simulation problem. In this work we investigate various neural network architectures and will attempt to predict the state of a quantum system in the following time step based on the current state.

We first examine and visualize multiple activation functions and test the effect on predictions. We then use Convolutional Neural Networks to find the ground state of a *Transverse Field Ising Model* by minimizing the energy, where the system has been parameterized for the paramagnetic and ferromagnetic phases, with near critical value for the transverse field in the ferromagnetic case. We then use Neural Ordinary Differential Equations to predict the time evolution of the system after an abrupt quench of the transverse field and evaluate the difference between different network types.

We initially split the simulation problem into two parts: Finding the ground state and predicting the time evolution. After that we combined both problems to produce a solution that results in a prediction of the future state of a *Transverse Field Ising Model* after a parameter quench of the *Hamiltonian*.

Contents

Part I.

Introduction and Background Theory

1. Introduction

The study of quantum many body systems pose many challenges, and the heart of the problem lies the exponential complexity of the many-body wave function.[1][2] The Hilbert space of a quantum system grows exponentially with the number of particles it contains, thus parametrizing a generic quantum state of N particles requires an exponential number of parameters.[3] Studying quantum many-body models is however crucial to understanding physical systems and their properties. In his work Stinchcombe gave an overview of systems that can be approximated – or at least be usefully described – by the spin-$\frac{1}{2}$ Ising model in a transverse field[4]. It has been shown that *toy* models such as the Transverse Field Ising Model (TFIM) can be used to find properties of physical systems. Examples of which are its use as a model for hydrogen bonded ferro-electrics[5][4], and its application to order-disorder ferro-electrics[4][6].

1.1. Motivation

Progress in computer science, particularly in the domain of machine learning has allowed the use of neural networks for approximating and solving many problems. These algorithms were proven to be capable of image generation [7], segmentation[8], playing games [9] and in controlling robots[10]. The application of artificial neural networks has also been expanded to the study of physics[11], and has been used as a representation for quantum states[1].

In the application of artificial neural networks to the problems presented by quantum dynamics, there is no review on the effect of using different activation functions. The non-linearity introduced through the use of activation functions in neural networks can have an impact on the performance of the network and on its capability to represent different transformations (see [12]). Moreover, the introduction of novel machine learning algorithms such as neural ODEs, which can be interpreted as the continuous form of a recurrent neural network, make them a very interesting test candidate for predicting the time evolution of quantum systems.

1.2. Our Contribution

In this work, we present a broad review of complex activation functions and assess their usefulness (or lack thereof) when used in conjunction with Convolutional neural networks

(CNNs) to solve one of the problems presented by quantum mechanics, namely, estimating the ground state of a TFIM system. We also investigate the use of regular Neural Ordinary Differential Equations (NODEs) and Trigonometric (Polynomials) Non-Autonomous Neural Ordinary Differential Equations (T-NANODEs) in estimating the time-evolution of quantum many-body systems without the use of *Monte Carlo* methods.

2. Quantum Mechanics

2.1. The Schrödinger Equation

One of the postulates in quantum mechanics is the concept of particle-wave duality and the wave function Ψ, which is a fundamental object in quantum mechanics and possibly the hardest to grasp in the classical world.[1]

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi \tag{2.1}$$

Equation 2.1.: The Time Dependent Schrödinger Equation

A wave function, which we get by solving the **Schrödinger equation**, contains all the information on a quantum state.[1][13] The size of the solution space however increases exponentially, and in principle an exponential amount of information is needed to fully encode a generic many-body quantum state.[1] This equation is however of little use to us in this body of work, and in the following section, we will show that it can be used in a different form that will make it more useful to us.

2.2. The Time Independent Schrödinger Equation

The Schrödinger equation as it is stated in Equation 2.1 is difficult to solve, as the potential energy V is a function of time and position, while also Ψ – the wave function, is dependent on time and position. We can however use the separation of variables and other assumptions to make this equation take a form that we can use for the estimation of ground states.

Using the separation of variables, the time dependent Schrödinger equation (2.1) can be written as:

$$\Psi(x,t) = \psi(x)\phi(t) \tag{2.2}$$

For this separable solution we have:

$$\frac{\partial\Psi}{\partial t} = \psi\frac{d\phi}{dt}, \quad \frac{\partial^2\Psi}{\partial x^2} = \frac{d^2\psi}{dx^2}\phi \tag{2.3}$$

Which makes the Schrödinger equation have the form:

$$i\hbar\psi\frac{d\phi}{dt} = -\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2}\phi + V\psi\phi$$

$$i\hbar\frac{1}{\phi}\frac{d\phi}{dt} = -\frac{\hbar^2}{2m}\frac{1}{\psi}\frac{d^2\psi}{dx^2} + V \tag{2.4}$$

In Equation 2.4 the left hand side is only dependent on t and right hand side only on x[1] which implies that both values have to be constant.
Denoting E as the separation constant, we can write the left hand side of Equation 2.4 as:

$$\frac{d\phi}{dt} = -\frac{iE}{\hbar}\phi \tag{2.5}$$

This equation has a general solution of the form $C\exp(-iEt/\hbar)$ where C is a constant. We can then rewrite Equation 2.2 as:

$$\Psi(x,t) = \psi(x)e^{-iEt/\hbar} \tag{2.6}$$

Where the constant of the general solution to Equation 2.5 has been absorbed into ψ.
The right hand side of Equation 2.4 can similarly be written as:

$$E\psi = -\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V\psi \tag{2.7}$$

This equation is called the time independent Schrödinger equation where V is the potential, the total energy is called the *Hamiltonian*:

$$H(x,p) = \frac{p^2}{2m} + V(x) \tag{2.8}$$

Substituting $p \to \frac{\hbar}{i}\frac{\partial}{\partial x}$ in Equation 2.8 gives us the Hamiltonian operator $\hat{H} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(X)$ using that in Equation 2.7 gives us:

$$E\psi = \hat{H}\psi \tag{2.9}$$

2.3. Matrix and Vector Representations

The time independent Schrödinger equation (2.9) is an example of an Eigenvalue equation. [14] The Hamiltonian operates on an Eigenfunction giving a constant E, times the same function. For bound states many Eigenfunction solutions exist such that $\hat{H}\psi_i = E_i\psi_i$ where ψ_i is an eigenfunction and E_i is the corresponding eigenvalue.

[1] We can assume that the potential V only depends on x because we're interested in the static properties of a system, such as the ground state.

The Hamiltonian is a Hermitian Operator An operator P is defined as hermitian if its r, s matrix element has the property $P_{rs} \equiv P^*_{sr}$ and although non-hermitian Hamiltonians have been studied [15][16][17] where a more general constraint on Hamiltonian operators – namely space-time reflection symmetry ($\mathcal{PT}$-symmetry) – has been stated, we can show that for a certain class of Hamiltonians, this assumption still holds.
The Hamiltonian (energy) operator (2.8) is composed of a kinetic energy part $p^2/2m$ and a potential term $V(x)$. The potential term involves the distance coordinate 'operator' x, which can be defined as an operator that acts on a function $f(x)$ to produce another function $xf(x)$ which is a multiple of the original one. For the x operator:

$$x_{rs} \equiv \int \psi^*_r x \psi_s dx = \int (\psi_r x^* \psi^*_s)^* dx = \int (\psi^*_s x^* \psi_r)^* dx \equiv x^*_{sr} \tag{2.10}$$

Which shows that the x operator is hermitian.

For the momentum operator in the one-dimensional case $p = \frac{\hbar}{i}\frac{\partial}{\partial x}$ we can also show the same:

$$p_{rs} \equiv \frac{\hbar}{i}\int_{-\infty}^{\infty} \psi^*_r \frac{\partial}{\partial x}\psi_s dx = \frac{\hbar}{i}\left([\psi^*_r\psi_s]^{\infty}_{-\infty} - \int_{-\infty}^{\infty} \psi_s \frac{\partial}{\partial x}\psi^*_r dx\right) \tag{2.11}$$

For finite quantum systems $\psi^*_r\psi_s$ will be zero, because the wave function vanishes at $\pm\infty$, which gives us:

$$p_{rs} \equiv \left(-\frac{\hbar}{i}\int_{-\infty}^{\infty} \psi^*_s \frac{\partial}{\partial x}\psi_r dx\right)^* \equiv p^*_{sr} \tag{2.12}$$

And since the hamiltonian is the combination of both of these operators, this shows that for this class of hamiltonians, the hermitian assumption holds.

For a hermitian operator, it holds that the eigenfunctions are orthonormal, where $\langle u_i|u_j\rangle = \delta_{ij}$ and u_i, u_j are an eigenfunction of the operator. We can then represent a state vector ψ using a complete set of these orthonormal basis states by defining the components of a state vector as:

$$\psi_i \equiv \langle u_i|\psi\rangle \qquad\qquad |\psi\rangle = \sum_i \psi_i |u_i\rangle \tag{2.13}$$

For an operator O we can define a matrix element $O_{ij} \equiv \langle u_i|O|u_j\rangle$ such that:

$$O|\psi\rangle = O\sum_j \psi_j |u_j\rangle = \sum_j \psi_j O|u_j\rangle \tag{2.14}$$

Multiplying $\langle u_i|$ from the right gives:

$$\sum_j \psi_j \langle u_i|O|u_j\rangle \equiv \sum_j \psi_j O_{ij} \equiv (O\psi)_i \tag{2.15}$$

We know that an operator acting on a wavefunction gives a wavefunction and the above equation is exactly that.

$$\begin{pmatrix} (O\psi)_1 \\ (O\psi)_2 \\ \dots \\ (O\psi)_i \\ \dots \end{pmatrix} = \begin{pmatrix} O_{11} & O_{12} & \dots & O_{1j} & \dots \\ O_{21} & O_{22} & \dots & O_{2j} & \dots \\ \dots & \dots & \dots & \dots & \dots \\ O_{i1} & O_{i2} & \dots & O_{ij} & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} (\psi)_1 \\ (\psi)_2 \\ \dots \\ (\psi)_j \\ \dots \end{pmatrix} \tag{2.16}$$

Where O is the hermitian operator in matrix form – in our case the Hamiltonian matrix – and ψ is the wavefunction in vector form.

2.4. The Ground State Problem

Lattice models, also known as generalized Ising models are used in many areas of science. They are routinely applied to alloy thermodynamics, solid-solid phase transitions, magnetic and thermal properties of solids, and fluid mechanics.[18] The problem of finding the true global ground state of a lattice model, which is essential to all these applications, has remained one of the hardest problems to solve for large systems. The ground state of a lattice model determine the $0K$ phase diagram of the system.[18]

Finding the ground state of a quantum many body system is analogous to finding the eigenfunctions which result in the lowest eigenvalues for a particular hamiltonian. If we have a hamiltonian matrix for a small system we can simply use iterative methods to calculate the eigenvalues and eigenvectors of the hamiltonian matrix, which would give us the ground state. In Equation 2.9 finding the smallest eigenvalue and its corresponding eigenstate for H would fulfill the equation and the resulting energy E would be the ground state for this particular system.

This approach however, is not viable for large multibody systems as the size of the hamiltonian grows exponentially with the number of particles. For spin $1/2$ models for example the size of the hamiltonian would be in the order of $O(2^n)$. Pan and Chen have shown that the eigenproblem is bound – within a relative error 2^{-b} – by an arithmatic complexity of $O(n^3 + (n \log^2 n) \log b)$[19], which would be in the order of $\Omega(C^n + n^2 2^n \log b)$ for a hamiltonian of n particles.

2.5. The Transverse Field Ising Model

Transverse Field Ising Models (TFIMs) are a class of lattice models introduced in 1963 by de Gennes as a pseudo spin model to describe the tunneling of protons in ferroelectric crystalls.[20][5] The TFIM describes a chain of spin particles with $S = \frac{1}{2}$ interacting through a ferromagnetic exchange J along the x (or y) axis. A magnetic field is applied along the z axis.[20]
de Gennes used this model for hydrogen-bonded ferroelectrics, such as KH_2PO_4, where the proton sits in one or other minimum of a double well, the transverse-field term represents the ability of the proton to tunnel between two minima (corresponding to $S^z = \pm\frac{1}{2}$).[4] The model hamiltonian is given by:

$$H = -h\sum_i S_i^x - \frac{1}{2}\sum_{ij} J_{ij} S_i^z S_j^z \tag{2.17}$$

Where h is the magnitude of the transverse field and J is an exchange interaction and the sum i runs over all lattice sites.[20][4]

3. Machine Learning

In this section we will introduce various concepts of machine learning. We will discuss the different methods through which a machine can learn, the models used, and the common problems which these methods can solve. In [21] a more general review of these methods were represented, in this work we will focus on the techniques that are relevant to the implemented solution.
A machine can generally learn by two methods: Learning from data and learning by interaction.[21] When a model attempts to learn from data, this typically would fall into the categories of supervised and unsupervised learning, whereas learning by interaction is a property of reinforcement learning. In this work we will use supervised learning to find both the ground state of a many body model and to predict the time evolution of that ground state after changing the parameters of the Hamiltonian.

3.1. Structure of a Neural Network

A neural network is typically structured into layers. The very first layer towards the input is called the input layer, while the last layer is called the output layer. Between the input and output layers, a number of layers are typically present, and these are called the hidden layers.
A neural network layer can be of various types, the most well known of which are: Fully Connected (Dense) layers, Convolutional layers and Recurrent layers.

3.1.1. Fully Connected Layers

Fully connected or Dense layers, are layers that perform a linear transformation of input data of size n to size m such that $f : \mathbb{R}^n \to \mathbb{R}^m$ for complex weighted networks this transformation can also act on a complex valued input such that $f_c : \mathbb{C}^n \to \mathbb{C}^m$. This can be represented in matrix form as

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix} \tag{3.1}$$

where the vector $\mathbf{b} \in \mathbb{R}^m$ is a bias vector. This type of layer is called Fully Connected because the ”neurons” in this layer are all connected to all the input variables – also called

input neurons, where a single output y_i is defined as $y_i = \sum_{j=1}^{n} w_{ij}x_j + b_i$ which is a linear combination of the inputs in addition to a bias variable.

3.1.2. Convolution Layers

A convolution layer is a type of layer where a "filter" is used to calculate the output values. A filter is type of shared weight matrix that acts on an input that is also typically shaped as a matrix $X \in \{\mathbb{R}^{N\times M}, \mathbb{C}^{N\times M}\}$. This is similar to a dense layer in that the output is a linear combination of the input, but a convolution layer shares the weights.

3.1.3. Activation Functions

As can be seen from 3.1.1, these layers can approximate linear functions, but to represent non-linear relations, an activation function has to be used. This introduces a non-linearity to the network, turning neural networks into a universal approximator[22].

3.1.4. Backpropagation

Finding an optimal set of parameters for a neural network can be difficult and research into this problem dates back to the early days of neural networks (see [23]), but through the introduction of the back-propagation (originally called *generalized delta rule*) algorithm in 1987, it has become possible to at least find a set of parameters that will reach a local minima in the solution space.

The *delta* rule mentioned by Rumelhart and McClelland, which was based originally on a method to tune adaptive machines[24], gives a basis for adjusting the weights of a network based on the error measured at the output. The *delta* rule can be derived as follows: Assuming that we use the square error of output o relative to a target value t

$$E = \frac{1}{2}(t - o)^2 \tag{3.2}$$

Where the output is the sum of weighted inputs with linear activation

$$o = \sum_j w_j i_j \tag{3.3}$$

Considering only one connection, the change of the error with respect to the weight connecting input i to the output o is then given by

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial w_j} = -(t - o)i \tag{3.4}$$

Since we are trying to minimize the error, weights should be changed so as to result in a negative change in the error (gradient descent), we therefore would have the following

condition on the change in weight

$$\Delta w_j \propto (t - o)i \tag{3.5}$$

Using a proportionality constant η, we thus arrive at the delta rule

$$\Delta w_j = \eta(t - o)i \tag{3.6}$$

A generalization of this derivation for the delta rule (hence the name *generalized delta rule*) is the back-propagation algorithm, instead of considering linear activation, we can use any function that is differentiable and non-decreasing [23], and propagate the error through each layer to update the network parameters using the chain rule

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial f(i)} \frac{\partial f(i)}{\partial i} \frac{\partial i}{\partial w_j} \tag{3.7}$$

where f here is an activation function, and i is the incoming input from the previous layer where $i = \sum_j o_j w_j$
We can thus recursively update the network parameters based on an error measurement using this gradient, which with a well-tuned learning rate η should at least lead to reaching a local minima.

3.2. Learning from Data

3.2.1. Ground State Problem

In supervised and unsupervised learning the network would typically have input data which are labeled, i.e. we would have a way of estimating the validity of the output data based on the inputs to the network, in the context of finding the ground state of a TFIM, we would calculate the energy of the model, where the energy is given by

$$E = \langle \psi | H | \psi \rangle \tag{3.8}$$

that would give us an objective function that the network should minimize. As a result of this equation, each prediction of the network can be assigned an objective value – in that case the energy, which we can use as an error value to propagate backwards through the network.

3.2.2. Time Evolution Problem

For the time evolution, the process is slightly different, the learning takes place based on samples from the analytical solution. This results in a data-set of "time" labeled state vectors $|\psi_t\rangle$ which are then used to estimate the error of the network using the overlap between the prediction and the true value (see equation 3.9).

$$Overlap = |\langle \psi_{pred} \,|\, \psi_t \rangle|^2 \tag{3.9}$$

This value for normalized (magnitude 1) vectors cannot be larger than 1, we can then denote an error as

$$L = 1 - |\langle \psi_{pred} \,|\, \psi_t \rangle|^2 \tag{3.10}$$

This error is then backpropagated through the network to estimate the gradients.

3.3. The Julia Programming Language

All of the implementations in this thesis have been written using the Julia Language [25] which is a high-level, human-readable and high performing language developed by Bezanson et al. from MIT. The Julia language attempts to solve the so-called *two-language problem*[25] where a high-level language (dubbed productivity language) that is easy to understand and which provides quick solutions for common scientific computing problems – such as: Matrix Multiplication, Array Operations and Vectorization – is used for prototyping, while a second language that is fast in tasks such as: Integer arithmetic, for loops, recursion and floating-point operations, like C++ or C, is used for the actual implementation.

Julia is able to combine performance and productivity in a single language due to a number of features that work well with each other[25][1]:

1. An expressive type system, allowing optional type annotations
2. Multiple dispatch using these types to select implementations
3. Metaprogramming for code generation
4. A dataflow type inference algorithm allowing types of most expressions to be inferred
5. Aggressive code specialization against run-time types
6. Just-In-Time (JIT) compilation using the LLVM compiler framework
7. A set of carefully written libraries that leverage the language design

The Julia language also provides different storage-types (data-types) for matrices that allow sparse matrices to be saved with memory complexity $O(n)$ where n is the number of non-zero elements, which a useful feature when dealing with very large multidimensional matrices, such as the ones used in Quantum Mechanics (recall that a Hamiltonian operator can be represented as a Matrix (see 2.3). Julia also provides functions that allow us to manipulate complex numbers out of the box by offering a data-type for complex numbers and a set of functions that operate on this data-type to extract for example the real part,

[1] This list of features is almost directly taken from the paper [25]

the imaginary part, the angle or the magnitude of a complex number. It is the presence of such high-level functions that enabled us to implement the complex activation functions presented in this work and to use them in building all of our models.

3.3.1. The Flux Machine Learning Library

In order to be able to use Julia for our purpose we needed a set of high-performance tools that can be used to train neural networks and in particular to perform automatic differentiation for ODE solvers, which would enable us to test NODEs in our experiments. These tools have been conveniently provided by FluxML[26][27] and DiffEqFlux[28] which is a Julia library for NODEs.

Flux is a machine learning framework that is built upon the Julia programming language and is built from the ground up to be simple and hackable[27]. Flux doesn't pick a specific level of abstraction (such as mathematical graphs or layer stacking) but instead, using careful design of the underlying automatic differentiation allows freely mixing mathematical expressions, built-in and custom layers and algorithms with control flow in one model.[27] Which makes Flux very easy to extend to new problems[27].

Existing machine learning frameworks achieve reverse-mode differentiation[29] by tracing (or partial evaluation), where a new tensor type is introduced that records all the basic mathematical operations performed, producing a graph with the control flow and data structures of the host language that is more easily differentiated than the original program.[27]

4. The Approximating Power of Neural Networks

The research into the expressive power of neural networks dates back to the 1980s[30] and the universal approximation theorem states that depth-2 networks can approximate any continuous function on a compact domain to any desired accuracy[30][31][32][33][34], indeed Hornik et al. has shown that feed-forward networks are capable of accurate approximation to any real-valued continuous function over a compact set[33]. This requirement holds whenever the inputs are bounded. His results have shown that the activation function need not be of any special kind, as long as they are continuous and non-constant, regardless of the dimension of the input space and regardless of the input space environment. In his work however, Hornik et al. did not investigate the rate at which the number of hidden layers must grow as the dimension of the input space increases[33].

In reality though, this expressive power is always limited by network size, Eldan and Shamir have shown that there is a trade-off between width and depth. In their paper [35] Eldan and Shamir have provided a simple 3-layer network that cannot be approximated by a 2-Layer network unless the width of the network grows exponentially. This type of results is referred to as depth efficiency of neural networks.[30] The limit to the expressive power is not only applicable to width but also to depth as pointed out by Lu et al. where the authors have shown that *width-bounded* networks are also limited, and that there is a family of **Re**ctified **L**inear **U**nit (RELU) networks that cannot be approximated by narrower networks whose depth increase is not more than polynomial[30].

4.1. Neural Ordinary Differential Equations

As we have mentioned earlier, a neural network can potentially approximate any continuous function, to any desired degree of accuracy regardless of the input size[1]. *Can neural networks then approximate the derivative of any differential function?*

To answer this question let us consider a differential equation of the form:

$$\dot{x} = \mathcal{F}(.) \tag{4.1}$$

[1] Assuming that the size of the network is not an issue

Where $\mathcal{F}(.)$ is an arbitrary function. Using the Euler method for this equation gives:

$$x_t = x_0 + \int \mathcal{F}(.)dt \tag{4.2}$$

This equation resembles a residual block in ResNets and establishes a connection between deep learning and differential equations.[36][37] In a broader sense, if $\mathcal{F}(.)$ were to be defined as a neural network, then using an ODE solver to integrate the function and solve the initial value problem, as one would with an ODE, this can be viewed as the continuous form of a ResNet, and is known as a Neural Ordinary Differential Equation (NODE).

A NODE is however not a universal approximator: Consider the function $f(x) = -x$, there is no ODE that can lead to a path from x_0 at $t = 0$ to $-x_0$ at time t, because the paths will intersect.[36] And what if the function $\mathcal{F}(.)$ is time-dependent i.e. $\mathcal{F}(t)$? Without having any information about the time, approximating this function would not be possible. In order to turn these networks into universal approximators, Dupont et al. suggested augmenting the input data with an extra dimension, and this dimension could be time (time-append). This results in a class of NODEs called Augmented Neural Ordinary Differential Equations (ANODEs). For an ANODE, the paths for the integrals can intersect because of the added dimension, however, this method does not add any explicit requirement to encourage the network to use the time input[36]. The solution to this problem is to make the weights of the network themselves a function of time. That is the core idea presented by Davis et al. in [36], where a class of NODEs is introduced that includes an explicit time-dependence for the weights of the neural network.

In our work we used this type of NODEs to solve the time evolution of quantum many-body systems with some success and in the following sections we shall present the methods used and results thereof.

Part II.

Methods

5. Complex Activation Functions

Because we are dealing with complex-valued neural networks, using a classical activation function would not be possible, it is therefore important to find activation functions that can act on a complex number, adding non-linearity similar to the real-valued networks.

5.1. Complex Differentiability

For a function f acting on a complex number z, with a real and imaginary parts x and y respectively, the Cauchy-Rieman equations provides a set of conditions under which this function is considered complex differentiable:

$$f(x, y) = u(x, y) + iv(x, y) \tag{5.1}$$

This function has to satisfy the set of equations 5.2 5.3 in order for it to be considered complex-differentiable.

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{5.2}$$

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \tag{5.3}$$

In literature one can find many examples of complex activation functions (see [39]). Many of the examined activation functions work by acting on the *real* and *imaginary* parts independently, and others add non-linearity using the phase angle. In the following section, different activation functions will be presented and visualized. It is to be noted that the activation functions presented here do not necessarily satisfy the Cauchy-Riemann equations and as such are not always holomorphic/complex differentiable.

It is also worth mentioning that all functions which are polynomial with complex coefficients, trigonometric or exponential are holomorphic and can indeed be used as activation functions. Despite that, their use in neural networks introduces complexity when calculating the gradients, it is the simplicity of the RELU activation function that made deeper networks possible.

5.2. Operation and Visualization of Activation Functions

In this work we have examined a total of 7 activation functions, many of which existed in previous works, and others are an extension of that. A complex activation function can be

considered to be a linear transformation of a complex number z acting on the phase θ and the magnitude m.

$$\begin{pmatrix} m \\ \theta \end{pmatrix}^T \cdot \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix} = \begin{pmatrix} \alpha m \\ \beta\theta \end{pmatrix}^T \tag{5.4}$$

Or equivalently.

$$f(z) = \alpha m \angle \beta\theta \tag{5.5}$$

With α representing the ratio between the input and output magnitudes and β the ratio between the input and output phase angles. We can then visualize complex activation functions in terms of the coefficients α and β.

5.3. Existing Activation Functions

5.3.1. zReLU

This activation function was proposed by Guberman (see [40]) and it works similarly to the scalar ReLU activation function, only the region where outputs are active is bounded by the *phase angle* instead of the sign of the input.

$$z\text{ReLU}(z) = \begin{cases} z & if\ \theta_z \in [0, \pi/2] \\ 0 & otherwise \end{cases} \tag{5.6}$$

Where θ_z is the phase angle of the input z. A visualization of this activation function can be seen in Figure 5.1. The $zReLU$ (as can bee seen in 5.1) only has a non-zero gradient in the first quadrant of the complex plane. That can potentially lead to a problem with vanishing gradients. This function is holomorphic as it satisfies the Cauchy-Rieman equations.

5.3.2. $\mathbb{C}$ReLU

This activation function was proposed by Trabelsi et al. (see [39]) and it acts independently on the Real and Imaginary parts of the input z.

$$\mathbb{C}\text{ReLU}(z) = \text{ReLU}(\Re(z)) + i\text{ReLU}(\Im(z)) \tag{5.7}$$

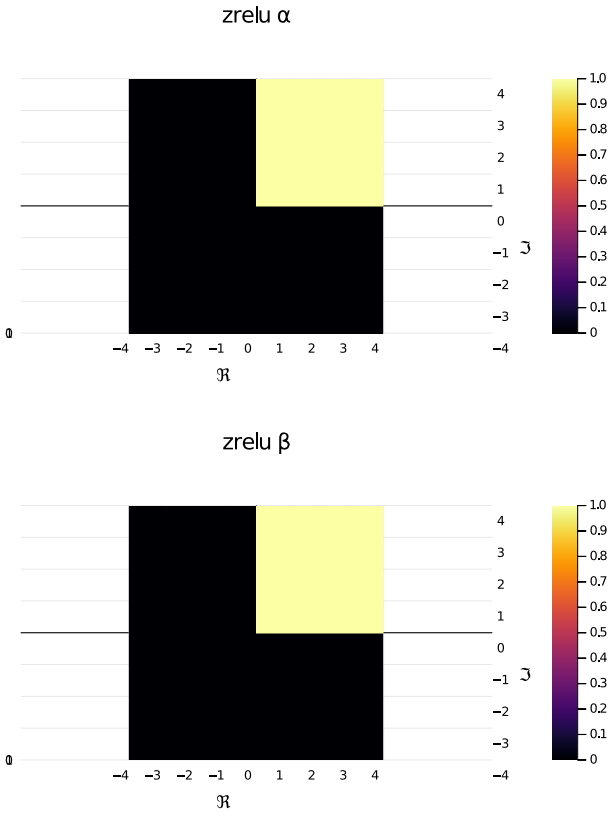

Figure 5.1.: Top: Magnitude coefficient(α) of the $zReLU$ activation function.
Bottom: Phase coefficient(β) of the $zReLU$ activation function

The $\mathbb{C}$ReLU function is not holomorphic

$$\mathbb{C}\text{ReLU}(x+iy) = \text{ReLU}(x) + i\text{ReLU}(y) \tag{5.8}$$

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 1 & if\ x \geq 0 \\ 0 & otherwise \end{cases} \tag{5.9}$$

$$\frac{\partial \text{ReLU}(y)}{\partial y} = \begin{cases} 1 & if\ y \geq 0 \\ 0 & otherwise \end{cases} \tag{5.10}$$

We can stop evaluating the Cauchy-Rieman conditions here as it is clear from the above equations that if the real part $\Re(z) = x$ is more than 0 while the imaginary part $\Im(z) = y$ is less than 0, the two partial derivatives would not be equal, thus proving that this activation function is non-holomorphic. However when both are strictly positive or negative, the $\mathbb{C}$ReLU function would be holomorphic.

In fact this result should hold for any activation function that acts independently on the real and imaginary parts, as the resulting function f would take the form

$$f(z) = f(x, y) = u(x) + iv(y) \tag{5.11}$$

and for the two functions u and y, the partial derivatives do not need to be equal, with the exception for the strictly positive or strictly negative case combined with a function that has a constant derivative.

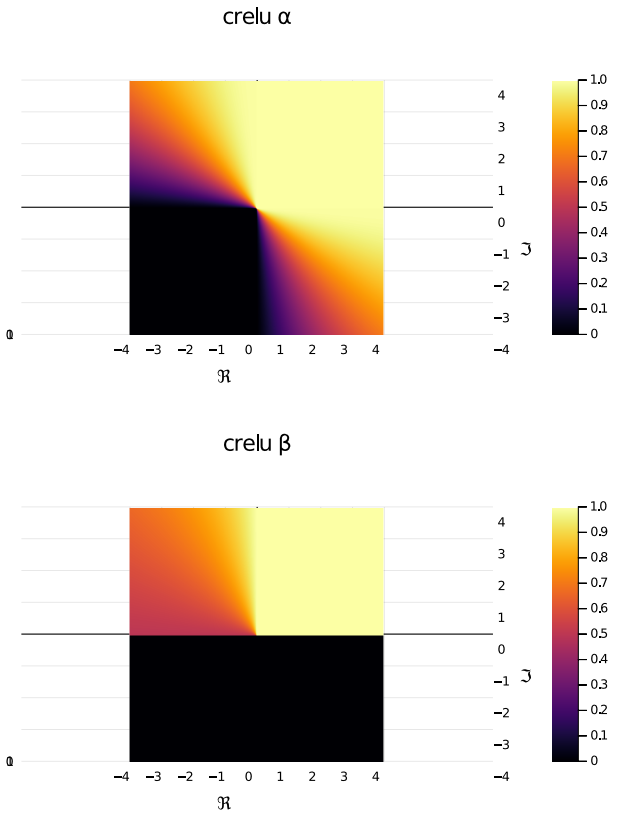

Figure 5.2.: Top: Magnitude coefficient(α) of the $\mathbb{C}ReLU$ activation function.
Bottom: Phase coefficient(β) of the $\mathbb{C}ReLU$ activation function

5.3.3. modReLU

This activation function was proposed by Arjovsky et al. (see [41]) and it is characterized by having a region around the origin where the activation output is zero, which is parameterized with the variable b.

$$\text{modReLU}(z) = \text{ReLU}(|z|+b)e^{i\theta_z} = \begin{cases} (|z|+b)\frac{z}{|z|} & \text{if } |z|+b \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

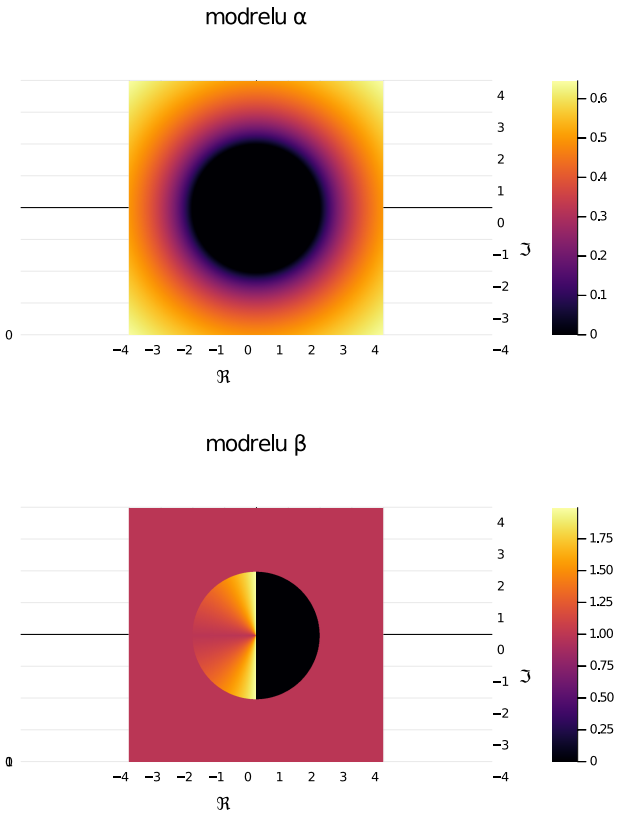

Figure 5.3.: Top: Magnitude coefficient(α) of the $modReLU$ activation function. Bottom: Phase coefficient(β) of the $modReLU$ activation function

The modReLU function is non-holomorphic

$$(|z|+b)\frac{z}{|z|} = (\sqrt{x^2+y^2}+b)\frac{x+iy}{\sqrt{x^2+y^2}} = \frac{x(\sqrt{x^2+y^2}+b)}{\sqrt{x^2+y^2}} + \frac{iy(\sqrt{x^2+y^2}+b)}{\sqrt{x^2+y^2}} \tag{5.13}$$

$$u = \frac{x(\sqrt{x^2+y^2}+b)}{\sqrt{x^2+y^2}} = \frac{x}{\sqrt{x^2+y^2}} + \frac{xb}{\sqrt{x^2+y^2}} \tag{5.14}$$

$$v = \frac{y(\sqrt{x^2+y^2}+b)}{\sqrt{x^2+y^2}} = \frac{y}{\sqrt{x^2+y^2}} + \frac{yb}{\sqrt{x^2+y^2}} \tag{5.15}$$

$$\frac{\partial u}{\partial x} = \frac{1+b}{\sqrt{x^2+y^2}} - \frac{1+b}{4(x^2+y^2)\sqrt{x^2+y^2}} \tag{5.16}$$

$$\frac{\partial v}{\partial y} = \frac{1+b}{\sqrt{x^2+y^2}} - \frac{1+b}{4(x^2+y^2)\sqrt{x^2+y^2}} \tag{5.17}$$

$$\frac{\partial u}{\partial y} = -\frac{x(1+b)}{4y(x^2+y^2)\sqrt{x^2+y^2}} \tag{5.18}$$

$$\frac{\partial v}{\partial x} = -\frac{y(1+b)}{4x(x^2+y^2)\sqrt{x^2+y^2}} \tag{5.19}$$

From equations 5.18 and 5.19 it is clear that this function doesn't satisfy the Cauchy-Rieman equations.

5.4. Proposed Activation Functions

5.4.1. $\mathbb{C}$swish

This activation function is similar to $\mathbb{C}ReLU$ (5.3.2) but instead of using a ReLU, we're using the swish activation function[12]. The swish activation function was not hand crafted, it was instead searched for using a combination of exhaustive and reinforcement learning-based search methods.[12]

The α and β graphs for this activation function can be seen in Figure 5.4.

$$\begin{aligned}\mathbb{C}\text{swish}(z) &= \text{swish}(\Re(z)) + i\text{swish}(\Im(z)) \\ &= \Re(z)\sigma(\gamma\Re(z)) + i\Im(z)\sigma(\gamma\Im(z))\end{aligned} \tag{5.20}$$

Where γ is a learn-able parameter (see [12]). This function is also non-holomorphic, except when both real and imaginary parts are large positive values; or small negative values (with a large absolute value), where the derivative of the function starts to become almost constant.

$$\frac{\partial(x\sigma(x))}{\partial x} = \sigma(x) + x\sigma(x)(1-\sigma(x)) \tag{5.21}$$

We can see that when x is large enough, this derivative reaches 1, and when x is a large negative value, this derivative will go towards 0.

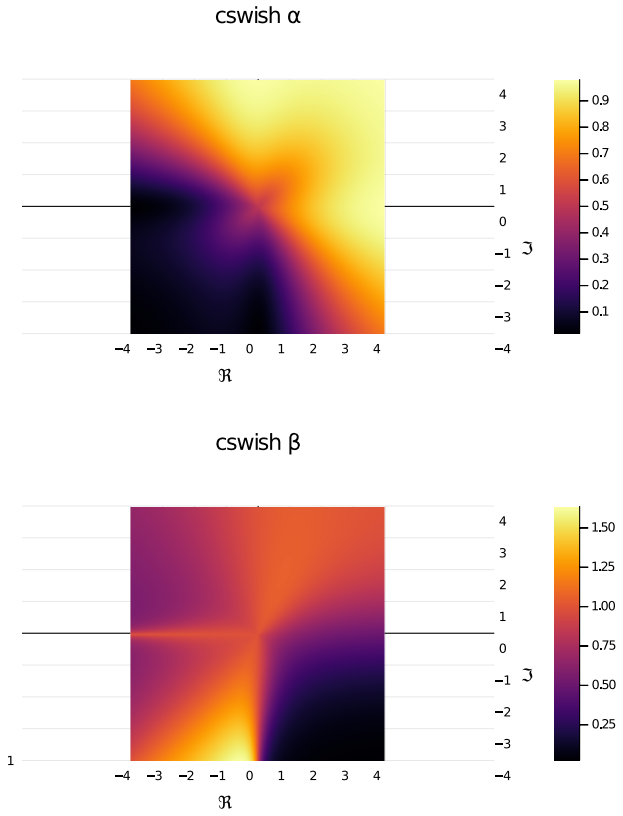

Figure 5.4.: Top: Magnitude coefficient(α) of the $cswish$ activation function.
Bottom: Phase coefficient(β) of the $cswish$ activation function

5.4.2. ℂleakyReLU

This activation function is the same as $\mathbb{C}ReLU$ (5.3.2) but it uses a *leakyReLU* instead. The *leakyReLU* has a non-zero gradient when saturated, and was first mentioned by Maas et al..[43][42]. The α and β graphs for this activation function can be seen in Figure 5.5.

$$\mathbb{C}\text{leakyReLU}(z) = \text{leakyReLU}(\Re(z)) + i\text{leakyReLU}(\Im(z)) \tag{5.22}$$

Similar to the ℂReLU case, this function is also non-holomorphic.

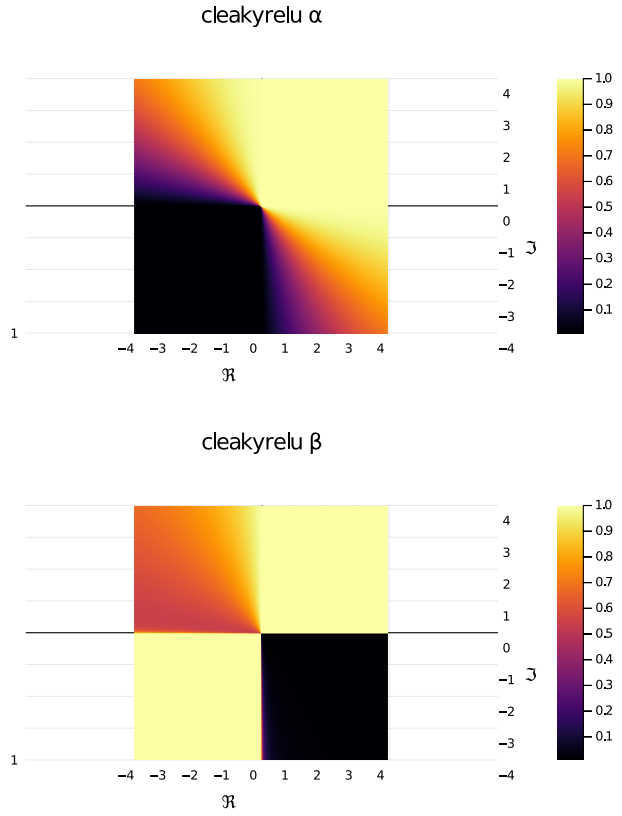

Figure 5.5.: Top: Magnitude coefficient(α) of the $\mathbb{C}leakyReLU$ activation function.
Bottom: Phase coefficient(β) of the $\mathbb{C}leakyReLU$ activation function

5.4.3. ℂCELU

A Continuously differentiable Exponential Linear Unit (CELU) was first proposed by Barron as a way to overcome one specific problem that an Exponential Linear Unit (ELU) has, which is that an ELU is not continuously differentiable if its shape parameter is not equal to 1.[44] The CELU activation function is defined as:

$$\text{CELU}(x,\alpha) = \begin{cases} x & if\ x \geq 0 \\ \alpha(exp(\frac{x}{\alpha} - 1) & otherwise \end{cases} \tag{5.23}$$

The $\mathbb{C}CELU$ is an extension of that for complex numbers exactly like in 5.3.2 or 5.4.1. Similar to the ℂswish function, this function is non-holomorphic, except for strictly large negative complex number or strictly positive ones.

$$\mathbb{C}\text{CELU}(z, \alpha) = \text{CELU}(\Re(z)) + i\text{CELU}(\Im(z)) \tag{5.24}$$

The α and β graphs for this activation function can be seen in Figure 5.6.

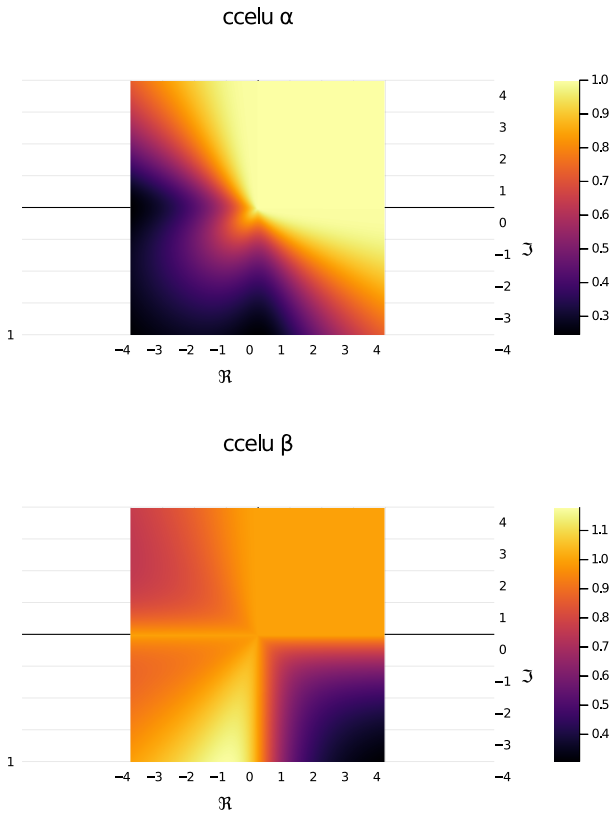

Figure 5.6.: Top: Magnitude coefficient(α) of the $ccelu$ activation function.
Bottom: Phase coefficient(β) of the $ccelu$ activation function

5.4.4. IzReLU

The zReLU (see 5.3.1) activation function has a very small domain where it is active, specifically only when $\theta_z \in [0, \pi/2]$, in this work we expanded the domain for this function by interpolating the output of second and fourth quadrant inputs.

$$\mathrm{IzReLU}(z) = \begin{cases} z & if\ \theta_z \in [0, \pi/2] \\ z\cos^2(\theta_z) & if\ \theta_z \in]0, -\pi/2] \\ z\sin^2(\theta_z) & if\ \theta_z \in]\pi/2, \pi] \\ 0 & otherwise \end{cases} \tag{5.25}$$

This results in a soft degradation towards 0 and increases the domain where the output is active, which might help avoid the problem of vanishing gradients, this makes the function however, non-holomorphic. The α and β graphs can be seen in Figure 5.7

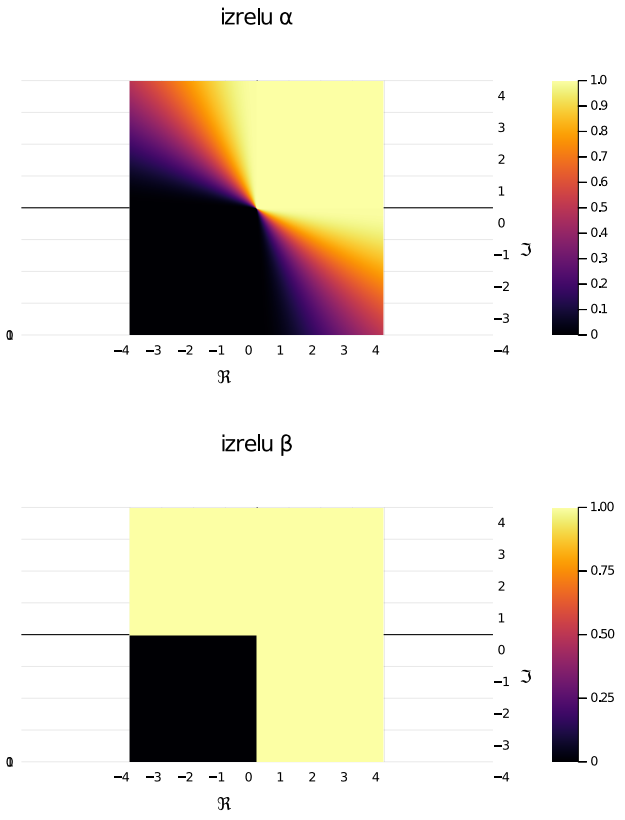

Figure 5.7.: Top: Magnitude coefficient(α) of the $izrelu$ activation function.
Bottom: Phase coefficient(β) of the $izrelu$ activation function

6. Method and Experiments

In this work, we have two parts for attempting to solve the quantum many body simulation problem – estimating the ground state, and predicting the time evolution. Both of these steps use different neural networks for the solution. In the next sections we shall present the architecture for these networks and the method used to attempt to solve these two problems.

6.1. The Ground State Problem

In solving this problem we parameterized the *Hamiltonian* of the TFIM system in two ways:

1. Paramagnetic, where the h parameter of the *Hamiltonian* is larger than the critical value.

2. Ferromagnetic, where h is less than (but near) the critical value.

After that we attempted to find the ground state of both systems using a convolutional neural network. Details about the implications of this parameterization can be found in the results section (see 8).

6.1.1. Network Architecture

CNNs have the benefit of weight sharing, which means that a relatively small network can find features in a very high dimensional problem. This property of CNNs makes them a very suitable architecture candidate for quantum many body problems, because the solution space is very large. The network might eventually learn to find a solution to the ground state problem in the portion of the Hilbert space where the solution lies.

The Network is constructed of three convolution layers and two fully connected layers as can be seen in Figure 6.1. The architecture has been kept relatively shallow in order to avoid the problem of vanishing/exploding gradients.
Throughout the network the same activation function has been used after every convolution layer and after every fully connected layer except the last layer which has an identity activation (linear). The kernels used are all of size 3x3 using same padding for each layer. A detailed description of the network architecture can be seen in Table 6.1

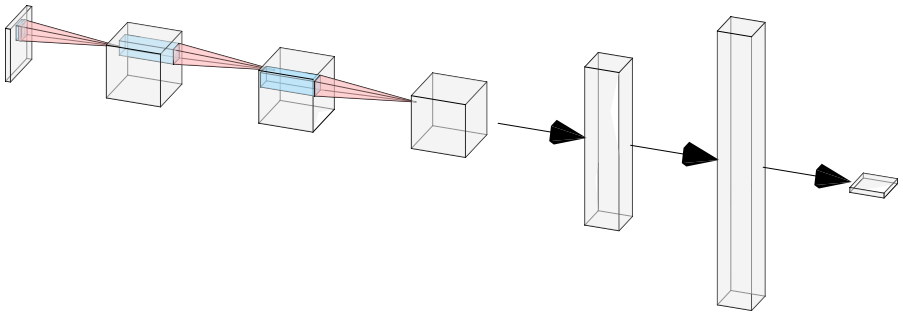

Figure 6.1.: Schematic diagram for the network used to find the ground-state

Layer	Type	# Parameters	Input Dimensions	Output Dimensions
1	Input Layer			$3 \times 3 (\times 512)$
2	3x3 Convolution, Stride = 1, Same Padding	30	$3 \times 3 \times 1$	$3 \times 3 \times 3$
3	3x3 Convolution, Stride = 1, Same Padding	84	$3 \times 3 \times 3$	$3 \times 3 \times 3$
4	3x3 Convolution, Stride = 1, Same Padding	84	$3 \times 3 \times 3$	$3 \times 3 \times 3$
5	Dense Layer	8400	$3 \times 3 \times 3$	300×1
6	Dense Layer	301	300×1	$1 \times 1 (\times 512)$

Table 6.1.: Detailed description of the Network used for the Ground State

6.1.2. Size of the Network Parameters

The model used has 8899 complex valued parameters. This was done by using convolutions on each one of the possible spin states instead of treating the entire spin configuration as one input, which when considering 9 Qubits would be 512 possible states.
The network has only one output neuron and for each spin configuration a complex value is produced, resulting in a vector that is as large as the number of possible spin configurations – in our case 512.

6.2. The Time Evolution Problem

For this problem we initialized the TFIM with a specific value for the *ferromagnetic exchange* $J = 1$ and another value $h = 5$ for the *transverse field*. We then perform a quench of the parameters for the *Hamiltonian* by setting $h = 0$ and used the time evolution network to predict model's evolution from the initial state to a another state Ψ_t under the changed conditions $h = 0$, the system therefore undergoes a phase transition from the paramagnetic phase to the ferromagnetic phase.

We have for this experiment two inputs: The initial state Ψ_0 and a hamiltonian H. This type of problem can be analytically solved using the equation for time evolution:

$$|\Psi_t\rangle = |\Psi_0\rangle\, e^{-\tau H} \tag{6.1}$$

where τ is the imaginary time it[1]. We have done this in two ways using NODEs: We used the entire state vector as a single input, which in our case would be a neural network with 512 input neurons; and we used a neural network with a single input neuron but augmented the parameters to add time-dependency to the weights – this approach is called Non-Autonomous, because the time dependency is forced and not learned.

6.2.1. Full State Model

Network Architecture

For full state time evolution, we initially tried using a convolutional neural network, however, the network couldn't predict the time evolution with a high enough accuracy. We therefore used a fully connected network with 512 input neurons; 2 hidden layers of sizes 1024 and 512 respectively; and 512 output neurons for the output layer.

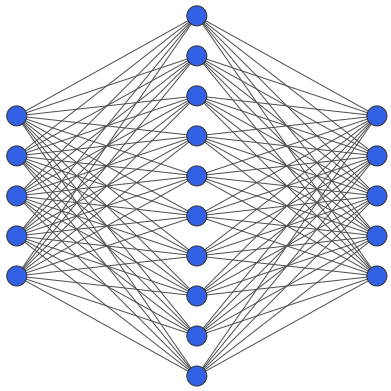

Figure 6.2.: Schematic diagram for the architecture of the time evolution network

Similar to the network in 6.1, this network also uses the same activation function after every layer, except at the output layer which has linear activation. A detailed description of the network used for the time evolution problem is show in Table 6.2.

Size of the Network Parameters

Because we didn't use a convolutional network like with the ground state problem this network has a large number of parameters (262,656).

[1] $i = \sqrt{-1}$

Layer	Type	# Parameters	# Inputs	# Outputs	Activation
1	Fully Connected	524288	512	1024	zReLU/$\mathbb{C}$ReLU
2	Fully Connected	524800	1024	512	zReLU/$\mathbb{C}$ReLU
3	Fully Connected	262656	512	512	Linear/None

Table 6.2.: Detailed description of the Network used for the time evolution

6.2.2. Using a Non-Autonomous Neural ODE

Network Architecture

Because of the large number of parameters involved in trying to predict the time evolution of an entire state of size 2^n we decided to use a Trigonometric (Polynomials) Non-Autonomous Neural Ordinary Differential Equation (T-NANODE) with only one input neuron and 50 hidden units. We *destructured* the network and then recombine it with an augmented set of parameters that are time dependent, using the trigonometric polynomials method of the 4th order – T-NANODE – mentioned in the paper by Davis et al. in [36] where the individual weights are dependent on time as given by the equation:

$$W_{t,ij} = a_0 + \sum_{n=1}^{d} a_n \cos(nt) + \sum_{n=1}^{d} b_n \sin(nt) \tag{6.2}$$

Where in our case $d = 4$, and the a_0 parameter has been absorbed into the original parameter such that

$$W_{t,ij} = W_{ij} + \sum_{n=1}^{d} a_n \cos(nt) + \sum_{n=1}^{d} b_n \sin(nt) \tag{6.3}$$

We also added a normalization layer $\mathbf{z} = \frac{\mathbf{z}}{||\mathbf{z}||}$ to avoid activations that are too large or too small after the first layer.

The network also uses the same activation function after every layer, except at the output layer which has linear activation. A detailed description of the network used for the time evolution problem is shown in Table 6.3.

Layer	Type	# Parameters	# Inputs	# Outputs	Activation
1	Fully Connected	900(800 for Time)	1	50	$\mathbb{C}$ReLU
2	$Norm(\mathbf{z}) = \frac{\mathbf{z}}{\|\|\mathbf{z}\|\|}$	0	50	50	N/A
3	Fully Connected	459(408 for Time)	50	1	None/Linear

Table 6.3.: Detailed description of the T-NANODE network

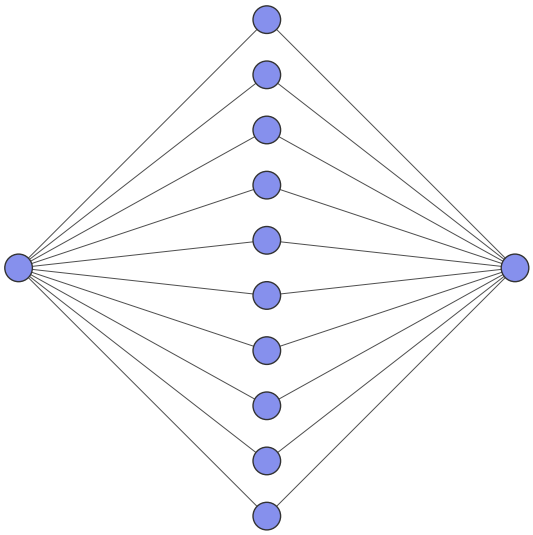

Figure 6.3.: Schematic diagram for the architecture of the T-NANODE network

Size of the Network Parameters

The resulting network has a total number of parameters of only 1359. This network was then trained to predict the next 4 time steps where $\Delta t = 0.01$ for each member of the state vector.

7. Loss Function and Training Method

7.1. Ground State Problem

7.1.1. Loss Function

For the ground state problem, we used the total energy of the system from Equation 3.8 as the loss function, i.e. the network will learn to minimize this function. There is no guarantee that the network will actually reach the true minimum, however, we used the overlap with the true ground state as a validation mechanism to keep the model with the best overlap. The exact loss function used is

$$L(\psi) = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle} \tag{7.1}$$

Where $\langle \psi | \psi \rangle$ is a normalization term because the generated state ψ is not normalized. For validation we used the overlap error $1 - |\langle \psi | \psi_0 \rangle|^2$.

7.1.2. Training

For this problem we used batch gradient descent with a Rectified ADAM (RADAM) optimizer. The network has been trained for around 2000 epochs, where each epoch is a full pass over all the possible spin configurations. We experimented with different activation functions for the network and experimental results showed that the $\mathbb{C}$ReLU and $\mathbb{C}$swish activation (see 5.3.2 and 5.4.1) functions seems to be better in terms of the final accuracy while also being much faster than other activation functions for this network.

7.2. Time Evolution Problem

For the time evolution problem we used a neural ODE approach [37] where the neural network instead of estimating the output state directly, it estimates the derivative of the state with respect to time. This type of network is analogous to a continuous-depth recurrent neural network [37] in that the number of ODE solver evaluations would be comparable to the time steps in a recurrent neural network.

7.2.1. Loss Function

For the loss function, we experimented with both the overlap error and the square absolute error. The network must also take into account the accuracy of evaluations in the time steps in between the end-time and start-time, to do this we calculated the sum of all the errors in the intermediate time steps, by specifying a particular time-interval where the network should save its output at. We used a combination of the sum of overlap and the square absolute error.

$$L(\theta) = \sum_{t=0}^{t} |\psi(t) - \psi_{pred}(t,\theta)|^2 + (1 - |\langle \psi(t) | \psi_{pred}(t,\theta) \rangle|^2) \tag{7.2}$$

Where θ represents the network parameters.

7.2.2. Training

For the first network we used batch gradient descent with a RADAM optimizer for at least 200 iterations and at most 2000 iterations. This large variation is due to the fact that the time for each iteration is not constant but depends on the number of solver evaluations. We used 51 input samples for this network when trying to predict the evolution of the state based on the time evolution of the Schrödinger equation, which in this case are the ground state of the TFIM model and 50 other state values for 50 discretized time steps in the future using the analytical solution of the time evolution.

For the case where we used a T-NANODE, we set the solver time-span to only $t = 0.0$ to $t = 0.04$ and collected the outputs in a single vector for each member of the state – this leads to at least 512 solver evaluations. The output is then formed into a matrix with the columns representing a state for each time-step $\Delta t = 0.01$[1]. We initially trained the model for 1000 iterations with a RADAM optimizer then used AdaBelief for another 1000 iterations.

[1]Note that the time-step here is not the solver time-step. The solver has a variable time-step but we explicitly required the result of the solver at this specific interval.

Part III.

Results and Conclusion

8. Results

One interesting property of the TFIM system is that it undergoes a quantum phase transition from a paramagnetic phase ($|h|> 1$) to a ferromagnetic phase ($0 < |h|< 1$).[45]

In our experiments with the ground state estimation, we used two different values for the parameter h: $h = 5$ where the system is in a paramagnetic phase; and $h = 0.75$ where the system is in a ferromagnetic phase and is incidentally close to the critical value at which the phase transition occurs (at $h = 1$). This near critical parameterization poses a challenge for neural networks as the correlation length of spin-spin interactions gradually increase with time (see [45]) making it harder to represent the wave function.

In the experiments with time evolution the TFIM undergoes a transition from the paramagnetic to the ferromagnetic phase, due to the parameter quench of the *Hamiltonian* $h = 5 \rightarrow 0$.

8.1. Ground State Estimation

The parameters that we used lead to a division into two different types of a TFIM based on the relative strength of the transverse field:

1. The strong field model (paramagentic phase) – the field parameter is larger than the interaction $h > J$ but not large enough to reach the strong field limit, where the particles are fully polarized (see [20]).

2. The weak field model (ferromagnetic phase) – the field strength is smaller than the interaction $h < J$ and the particles can interact almost freely, but not enough to reach the zero field limit (see [20]).

The corresponding Hamiltonian for both of these of these models was used to calculate the estimated energy based on the network output, and the true ground state was found using eigendecomposition of the Hamiltonian, where the smallest eigenvalue corresponds to the ground state of the model.

8.1.1. Strong Field Model

In this model the interaction between the spin particles have a smaller influence on the resulting ground state and as a result, the wave function Ψ in the ground state becomes

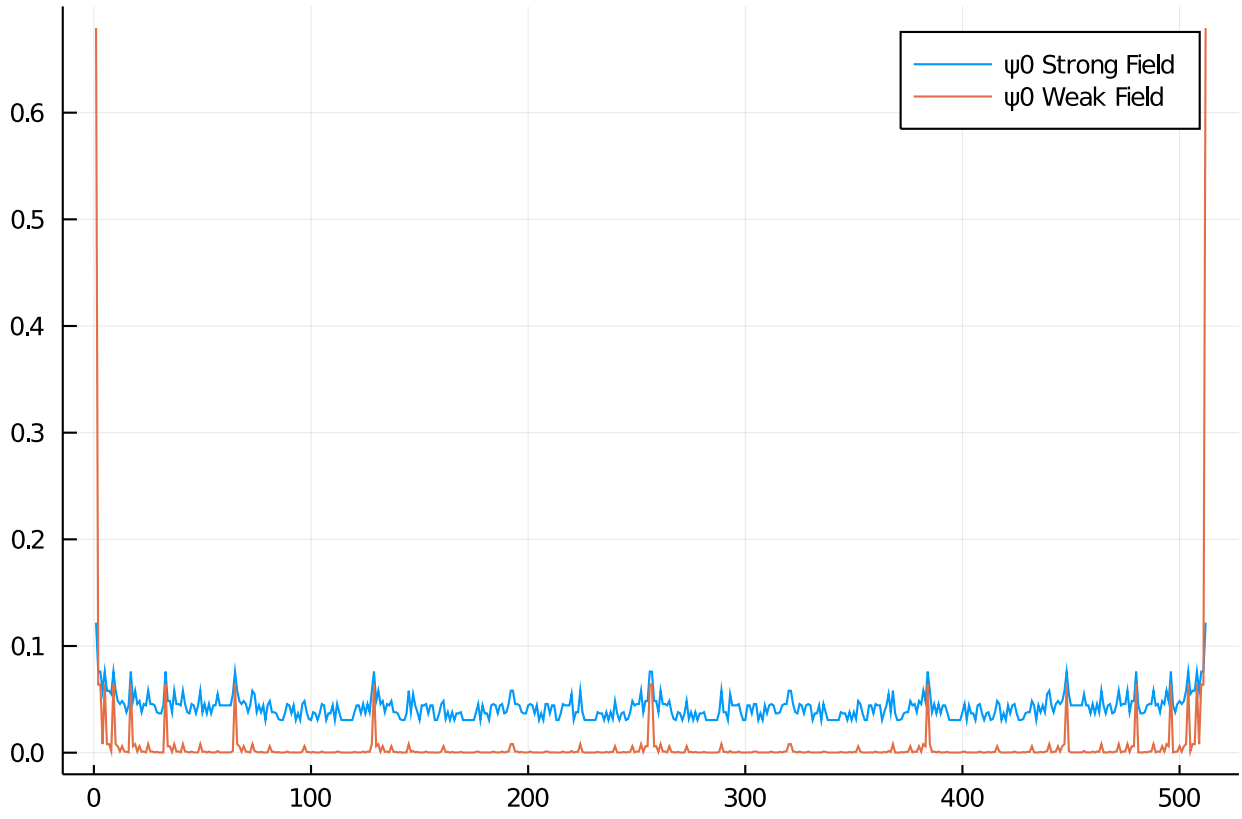

Figure 8.1.: Wave function of a weak field model versus that of a strong field model

slightly more uniform in that the components of the state vector are not negligible in relation to each other. The network that was created is able to reach a ground state prediction that is very close to the true ground state. We have tested all 7 of the activation functions. Details to each can be seen in the appendix, the best result can be seen in Figures 8.2 and 8.3.

8.1.2. Weak Field Model

In this model the interaction between the spin particles have a relatively large influence on the resulting ground state – due to a higher correlation length – and as a result, the wave function Ψ in the ground state becomes non-uniform in that the components at the edges of the state vector are large relative to the other components. Just like the strong field model, the network is able to reach a ground state prediction that is very close to the true ground state, however the overlap error didn't decrease beyond 50%, this might be because the remaining 50% of the overlap accounts for less than 0.023% of the ground

state energy, which results in very small gradients. We also noticed that the state with the lowest overlap error doesn't necessarily map to the lowest energy state, this suggests that the relation between the overlap error and the energy is not linear.

As with the previous section 8.1.1, we tested all 7 of the activation functions. The best results can be seen in Figures 8.4, 8.5, 8.6 and 8.7.

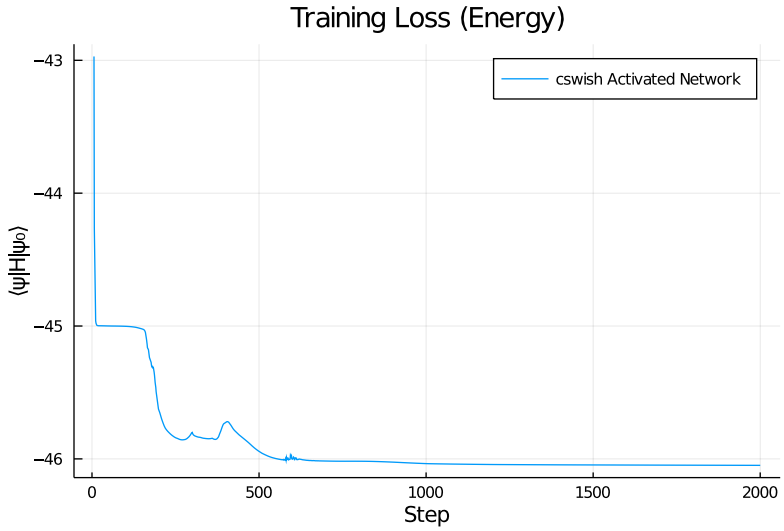

Figure 8.2.: Predicted ground state energy of the best network for the strong field model. The network uses the $\mathbb{C}$swish 5.4.1 activation function and was trained for 2000 iterations. The true minimum is -46.05247819 and the network reached a minimum of -46.04873275756836.

8.2. Time Evolution Problem

For the time evolution problem we used a Neural ODE approach where the neural network is considered to be a representation of the derivative of the state vector with respect to time. An ODE solver is then used to solve this problem and produce a series of outputs for the given time span. We tested both convolutional and fully connected networks for full state evolution, and have found that convolutional networks (1-D Convolution) don't generalize well in this case so we opted for the fully connected type.

8.2.1. Predicting Full State Evolution

After training for about 500 iterations on the time span from $t = 0$ to $t = 0.5s$ (divided into discrete time steps with 0.01 seconds each) the NODE network was able to

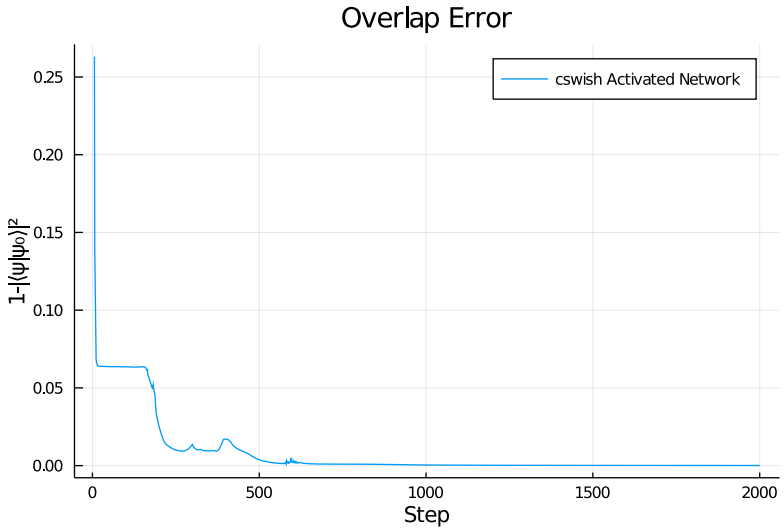

Figure 8.3.: The overlap error of the best network for the strong field model. The network reached a minimum overlap error of 7.4×10^{-5} which corresponds to a minimum energy of -46.04873123677644.

reach a total overlap error of 0.1 and a total loss (see equation 7.2) of 0.21. The network was then modified (by changing the time span of the solver) without any additional training to predict the time evolution up to time $t = 1$. The results can be seen in Figure 8.8.

Training with this type of network was prohibitively time consuming as each iteration can take a long time, to solve this we can reduce the tolerances of the solver, however this results in less accurate predictions and sometimes can lead to an unstable performance. For this reason we couldn't test all 7 activation functions and only solved the time evolution starting from the strong field state (see 8.1.1).

8.2.2. Using a Non-Autonomous Neural ODE

After training for more than 2000 iterations with the T-NANODE model, the network was able to predict the next 4 time-steps (each 0.01) with a maximum overlap error of roughly 2×10^{-4}. In order to test if the network can generalize for future predictions, the same network was then asked to predict the time evolution for 10 time-steps instead of 4, and although the overlap error increased rapidly, it was still capable of staying below the error of no-action (where the state vector is held constant). This shows that the network is trying to steer the time-evolution into the correct direction even without any data. The results of this type of network can be seen in Figure 8.9 and 8.10.

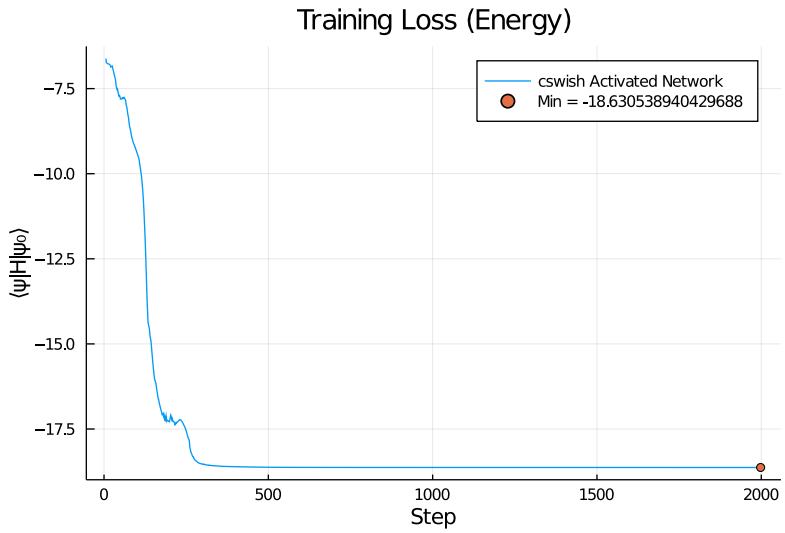

Figure 8.4.: The ground state energy predictions of the $\mathbb{C}$swish activated network for the weak field model, which reached the minimum overlap error. The true minimum is -18.63473128.

8.3. Combined Result

As was seen in the previous parts, we could predict the ground state and the time evolution for future time-steps (using T-NANODE) and as a result of that, it is possible to combine both networks to predict the time-evolution of a TFIM with a particular set of parameters after a quench of the transverse magnetic field (setting the parameter h to 0).

The combined models were able to find the ground state and to predict the time evolution for a time-span of 10 time-steps (as with the previous section) with an overlap error that also doesn't exceed the error reached individually. For the T-NANODE model this shows that the network has some resilience when it comes to small deviations from the inputs used during training. The results of this test can be seen in Figures 8.11 and 8.12

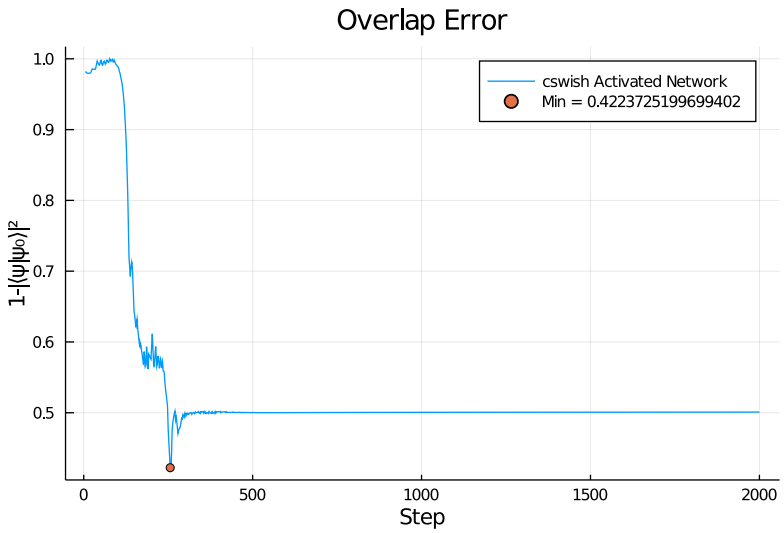

Figure 8.5.: The overlap error of the $\mathbb{C}$swish network for the weak field model, which reached the smallest overlap error. The smallest overlap error corresponds to an energy of -17.758975982666016.

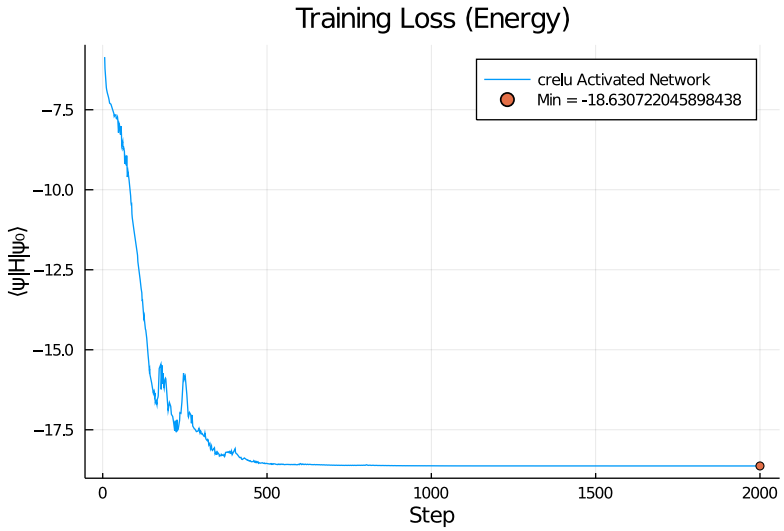

Figure 8.6.: The ground state energy predictions of the $\mathbb{C}$relu activated network for the weak field model, which reached the lowest ground state energy. The true minimum is -18.63473128.

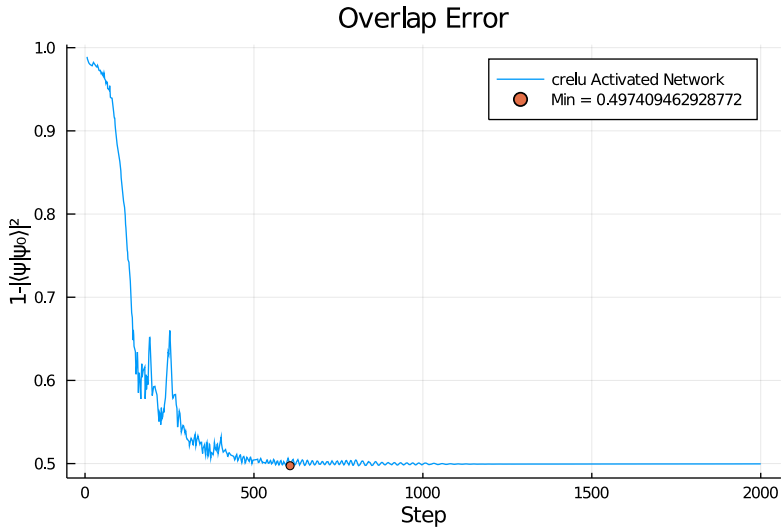

Figure 8.7.: The overlap error of the network that reached the lowest energy for the weak field model. The smallest overlap error corresponds to an energy of -18.630722045898438.

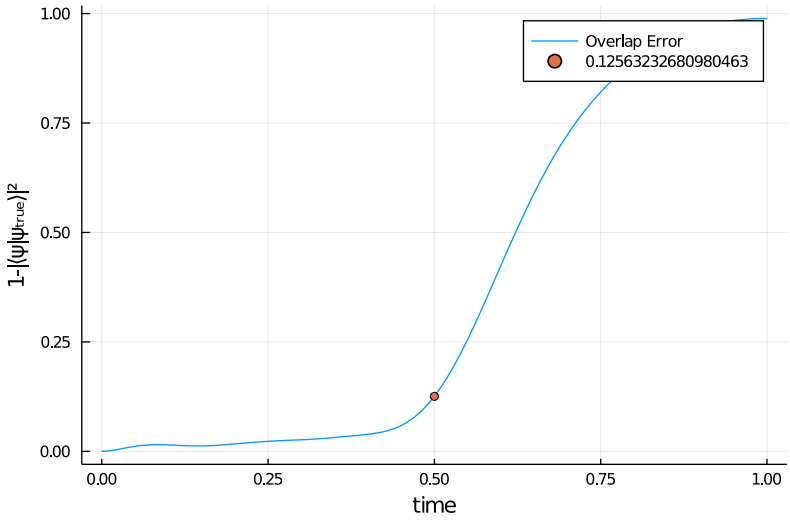

Figure 8.8.: Overlap error vs time for the NODE network used for full state evolution after a parameter quench of the Hamiltonian $h = 5 \rightarrow 0$

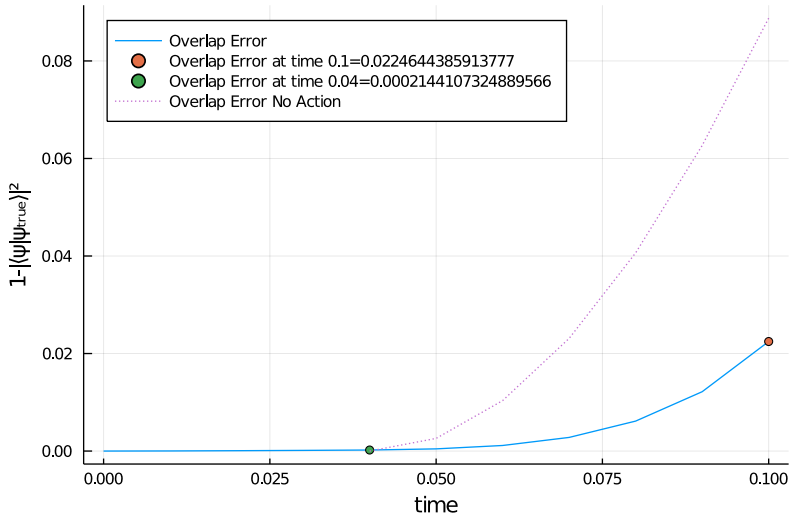

Figure 8.9.: Overlap error vs time for 10 time steps of the T-NANODE network after a parameter quench of the Hamiltonian $h = 5 \rightarrow 0$

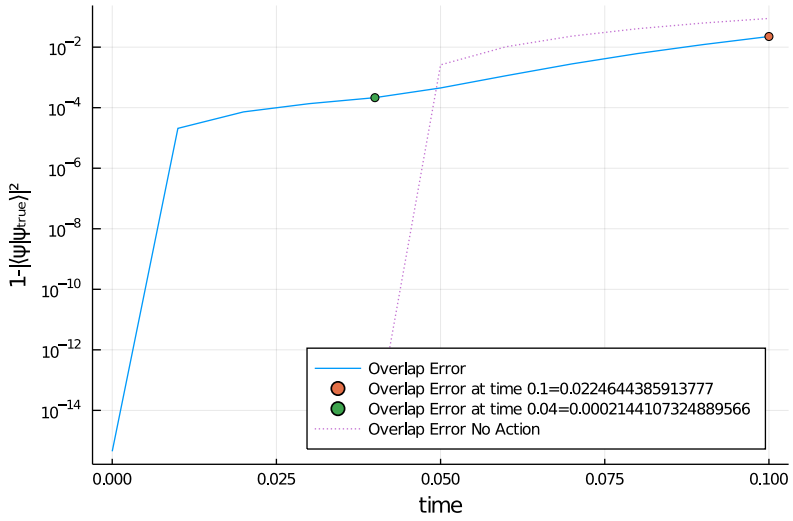

Figure 8.10.: Overlap error vs time for 10 time steps of the T-NANODE network in log-scale after a parameter quench of the Hamiltonian $h = 5 \rightarrow 0$

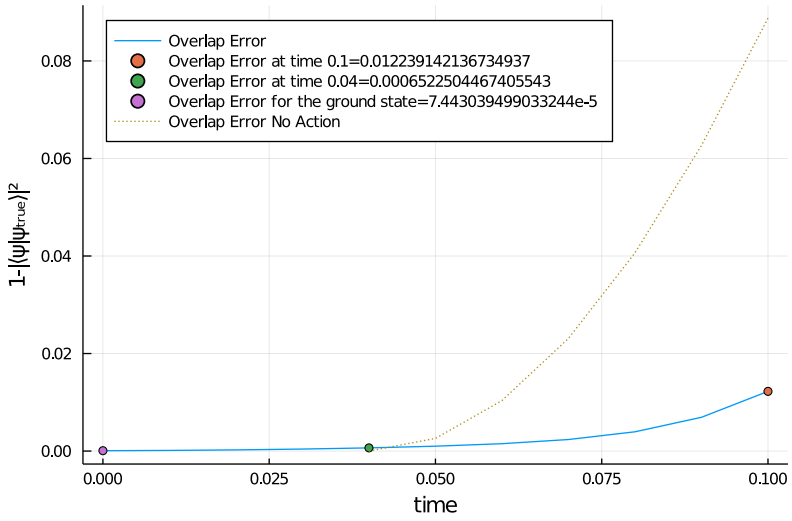

Figure 8.11.: Overlap error vs time for 10 time-steps of the combined solution after a parameter quench of the Hamiltonian $h = 5 \rightarrow 0$

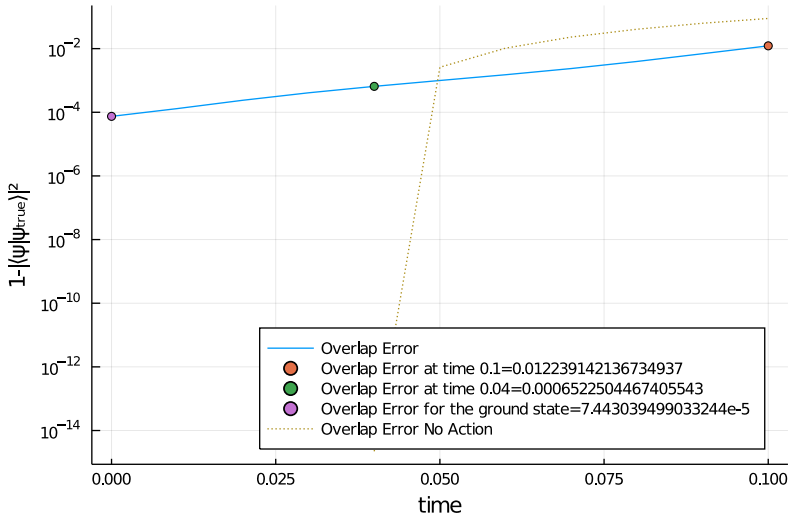

Figure 8.12.: Overlap error vs time for 10 time-steps of the combined solution after a parameter quench of the Hamiltonian $h = 5 \rightarrow 0$ in log-scale

9. Conclusion and Discussion

In this work we have shown that the ground state of a quantum many-body system (TFIM) can indeed be predicted just by minimizing the total energy of the system $E = \langle \psi \,|\, H \,|\, \psi \rangle$. We used in this approach standard machine learning methods with Complex-Valued weights and complex activation functions.

We also managed to train a network using a neural ODE approach to predict the time evolution of a TFIM for 50 time steps, where we reached a maximum overlap error of $\approx 12.6\%$, this result is not significant in comparison to other methods, however in our training strategy we used batch gradient descent and let the network learn for all 50 time-steps collectively. This makes it possible to generate a prediction – within an error – for more than just 1 time-step and can even modify the tolerances of the solver to either make the solution more accurate or less accurate in exchange for faster inference time.

We then extended the NODE approach and used a Non-Autonomous Neural Ordinary Differential Equation (NANODE) with trigonometric time dependency, which yielded satisfactory results and had a smaller number of parameters as our previous approach. This so-called T-NANODE network was able to predict the time evolution of 4 future time steps starting from the ground state prediction of the first network after a magnetic field quench $h = 5 \rightarrow 0$. The state overlap error between the predicted states and the true states did not exceed the overlap error reached when trained individually. In the following two sections we shall discuss in more detail the implications of our results.

9.1. Ground State Problem

In our results we have found that the ground state of a many body system can be well represented with standard machine learning techniques such as convolutional networks (and fully connected ones) in combination with complex activation functions, and that the network can possibly find a state that is very close to the true one, however this highly depends on the initial state of the system, and that this representation power (using convolutional layers) is sensitive to outliers in the output state. The usage of fully connected layers might have a better representational power but that comes at the cost of a large number of parameters.

9.2. Time Evolution Problem

In our results we only documented the models that were successful enough to be usable, but in our experiments we tested many types of NODEs: We used a 1-D convolutional network to model the time derivative of the system; we attempted using an ANODE (see [38]) and forcing the augmented dimension to be time by setting the output of the network used to 1 (this type was mentioned in [36]); we also attempted to implement a non-autonomous NODE[36] with a trigonometric time dependence (T-NANODE) and we trained another NODE using data generated from the analytical solution to the time-dependent Schrödinger equation.

In our experiments with ANODEs however, the solver calls took extremely long times (sometimes upwards of 2 hours) so we had to stop these experiments.

The most successful experiments were with standard NODEs when attempting to predict the time evolution of the entire state and the experiment with NANODEs by using the Julia Language [25] in combination with FluxML [27][26] to destructure the network and then recombine it with an augmented set of parameters. This suggests that it is possible to use NODEs particularly T-NANODEs for the simulation of Quantum Time Evolution, and that in combination with a suitable representation for the ground state it may be possible to simulate the dynamics of a particular many-body system.

9.3. Discussion

In our work we used standard neural networks and standard training, however it has already been shown that Restricted Bolzmann Machines (RBMs) can manage to represent Quantum States with high fidelity[1] and that they can be trained to predict unitary time evolution[1][46]. We might have been able to produce better results if instead of using traditional networks we had used RBMs. However, we focused instead on the use of NODEs for the prediction of unitary time evolution. It might be possible to represent this evolution using a RBM in combination with Ordinary Differential Equation (ODE) solvers instead of a regular neural network, which might have the potential to represent many-body quantum time evolution more effectively and might generalize better in different scenarios.

The methods used in this work separated the two problems into two different networks that were then combined to form the final result. Because of the flexibility of the Julia language and the framework built upon it – Flux – it would be very much possible to combine both the networks into a larger one, by adding the ODE solver calls as a layer inside this now bigger network. This method would potentially simplify training, however it might be harder to troubleshoot issues that arise during training such a network,

as the errors could be in either parts of the network. In the networks that we used, we also didn't ensure that the activation functions are holomorphic. The implications of this have also not been investigated further.

Mini-Batch Gradient Descent Since in these experiments we had an non-negligible number of inputs (related to the size of the state vector), we could potentially use *Mini-Batch* Gradient Descent for training both networks, which would make it easier to find a set of parameters that could optimize this problem more efficiently. The problem with this is that the prediction of the entire state vector needs to be normalized in order to be used along with the true state for the overlap-loss; or with the hamiltonian matrix for the energy (equations 3.10 and 7.1). That makes it difficult to find a meaningful normalization method for the individual inputs, unless we decide to use the square absolute error only for the time-evolution, and would force us to use the true ground-state of the TFIM for the ground state estimation. Alternatively, we could – for the time evolution – use the overlap error as a validation mechanism; or a mixture of both where the network is trained using a mini-batch with the square absolute error and then at the end of an epoch the normalized prediction would be used for a few iterations along with the overlap error. Whether or not this training regimen would be effective has not been investigated.

9.4. Outlook

In future work, it might be useful to investigate the use of NANODEs along with a neural network quantum state representation using RBMs, this would make the use of monte carlo sampling possible, the use of which in combination with NODEs hasn't been investigated. It is also conceivable that the use of 1-D convolutions for a NANODE might provide better results, as the spatial correlations between spin particles might be accounted for using a large enough filter.

For the specific case of T-NANODEs – which we used here, the effect of increased complexity (dimensions) for the time-dependent parameters; and whether there is a trade-off between the number of added time-dependent dimensions and the parameter size[1], can be investigated as a topic for future research.

[1] Similar to the width versus depth discussion in neural networks

Appendix

A. Additional Training Results for the Ground State Estimation

Note that we have included some additional activation functions that we haven't mentioned but have tested. These additional functions are:

1. wtanh: Which is defined as $\mathrm{wtanh}(x,y) = \frac{\tanh(x)}{(1-(x-3)e^{-x})} + \frac{i\tanh(y)}{(1-(y-3)e^{-y})}$, where x and y are the real and imaginary parts respectively. (Figures A.32 A.31 A.30 A.29)
2. sigmoid: The sigmoid function $\sigma(z) = \frac{1}{1-e^{-z}}$ (Figures A.48 A.47 A.46 A.45)
3. swish: Which is the original swish activation function defined as $\mathrm{swish}(z) = z\sigma(z)$ this function reached an energy value and an overlap error for the strong field model that is even smaller than the $\mathbb{C}$swish function. (Figures A.4 A.3 A.2 A.1)
4. lisht[47]: Defined as $\mathrm{lisht}(z) = z\tanh(z)$ (Figures A.44 A.43 A.42 A.41)
5. gelu[48]: Defined as $\mathrm{gelu}(z) = \frac{1}{2}z(1 + \tanh(\sqrt{\frac{2}{\pi}} * (z + 0.044715z^3)))$ (Figures A.40 A.39 A.38 A.37)

Note also that the sigmoid function ran into an optimization issue where the gradients were *NaNs*. It is also worth mentioning that the modReLU function can be parameterized, and that this parameter can be added as a learnable parameter to the network, where the optimizer can choose the best value to use every time a gradient is calculated. We did not investigate this possibility and settled for using a constant value for this parameter.

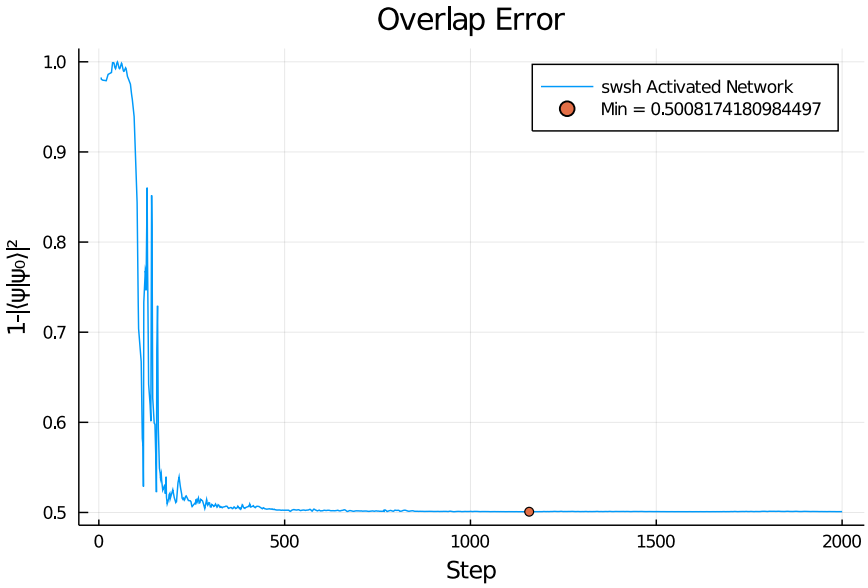

Figure A.1.: Training progress of the swish activated network: overlap vs step for the weak field model

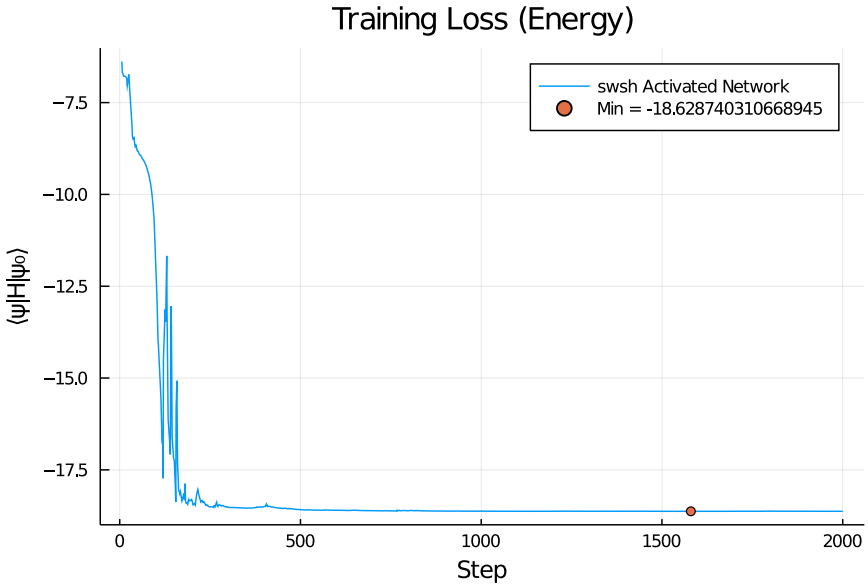

Figure A.2.: Training progress of the swish activated network: energy vs step for the weak field model

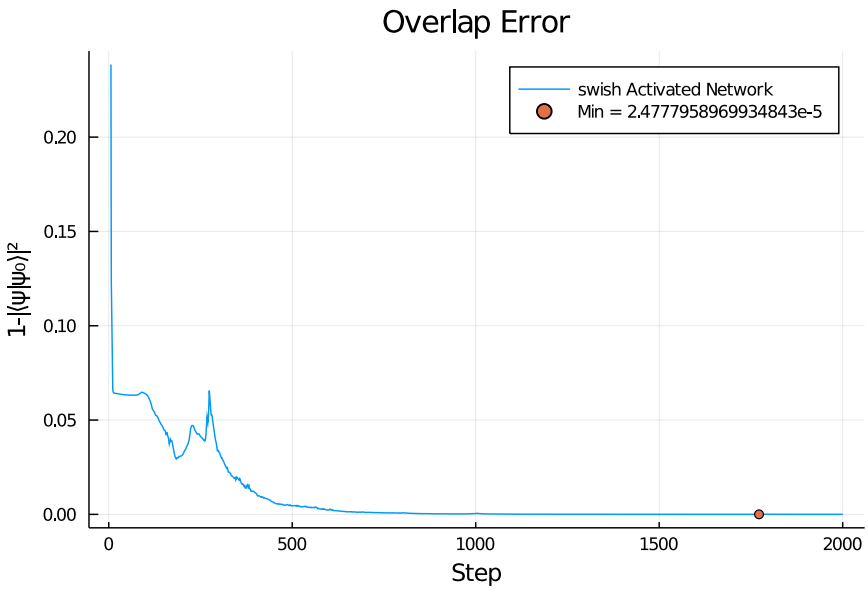

Figure A.3.: Training progress of the swish activated network: overlap vs step for the strong field model

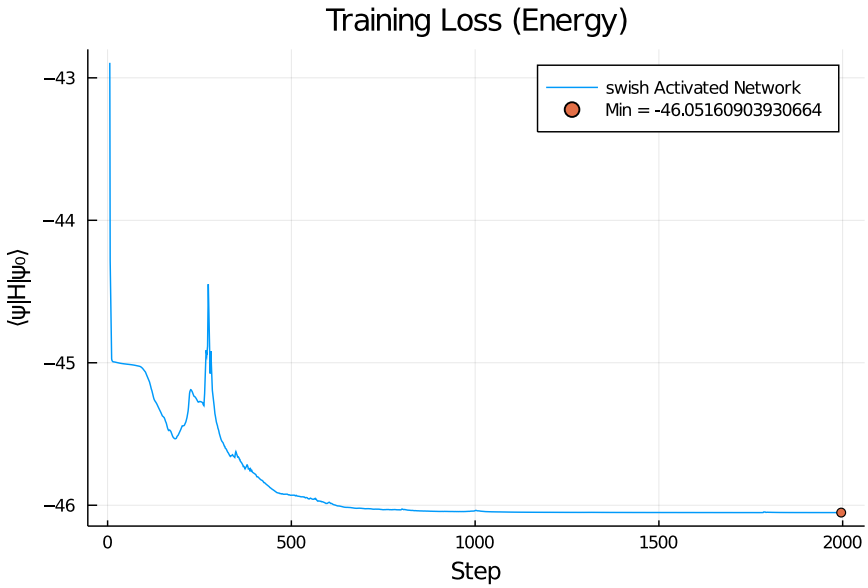

Figure A.4.: Training progress of the swish activated network: energy vs step for the strong field model

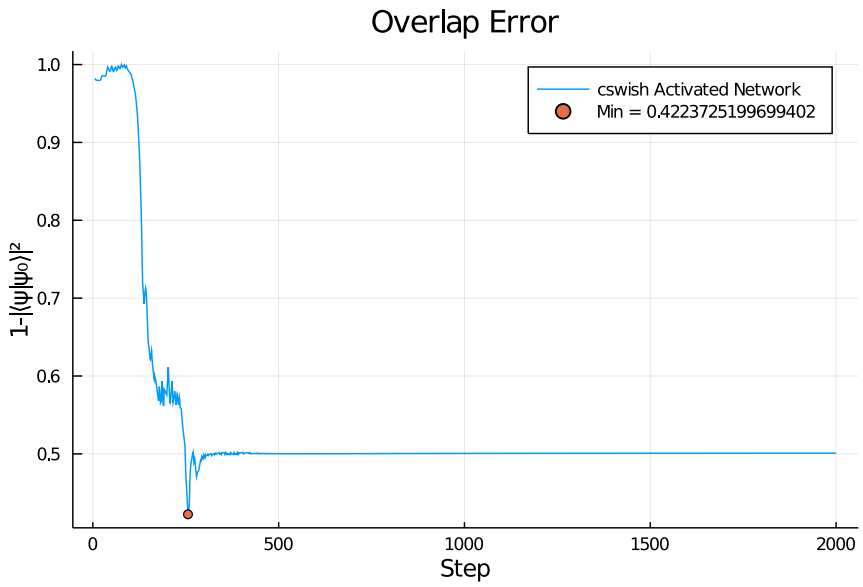

Figure A.5.: Training progress of the cswish activated network: overlap vs step for the weak field model

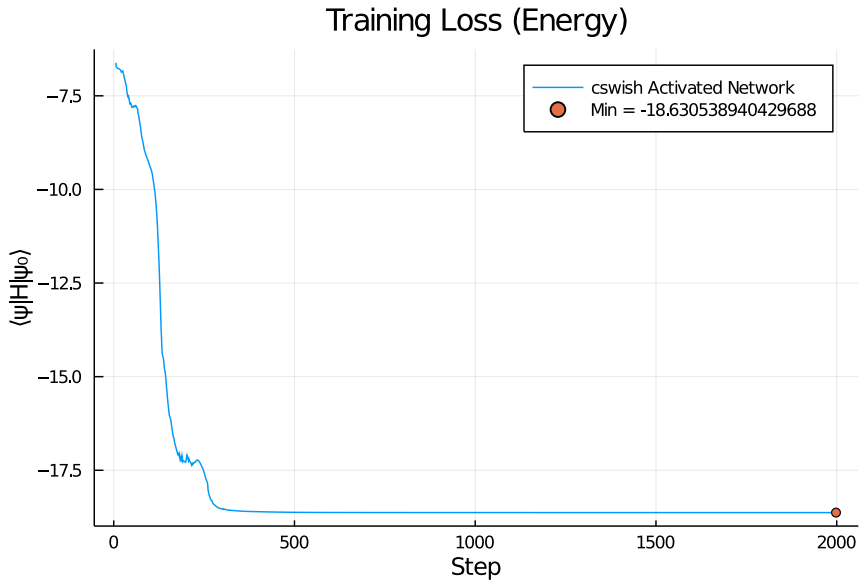

Figure A.6.: Training progress of the cswish activated network: energy vs step for the weak field model

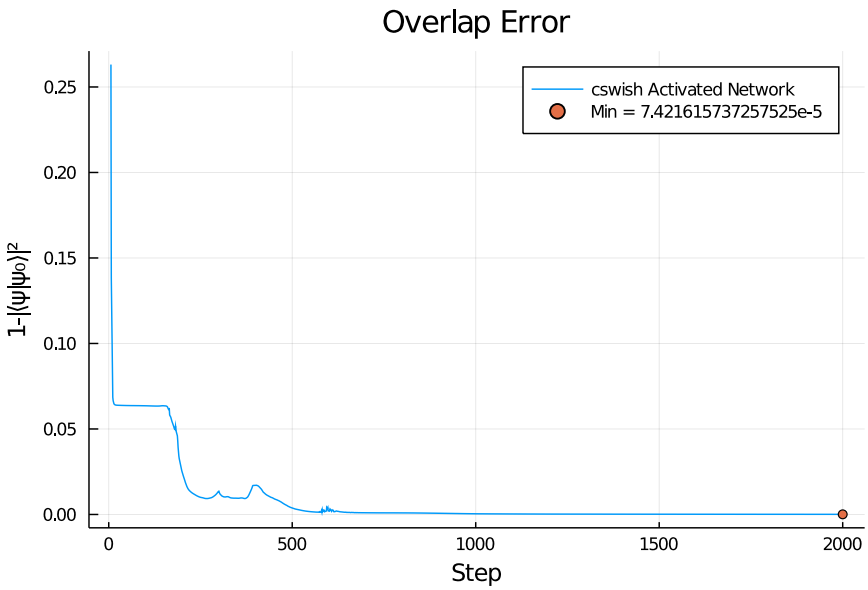

Figure A.7.: Training progress of the cswish activated network: overlap vs step for the strong field model

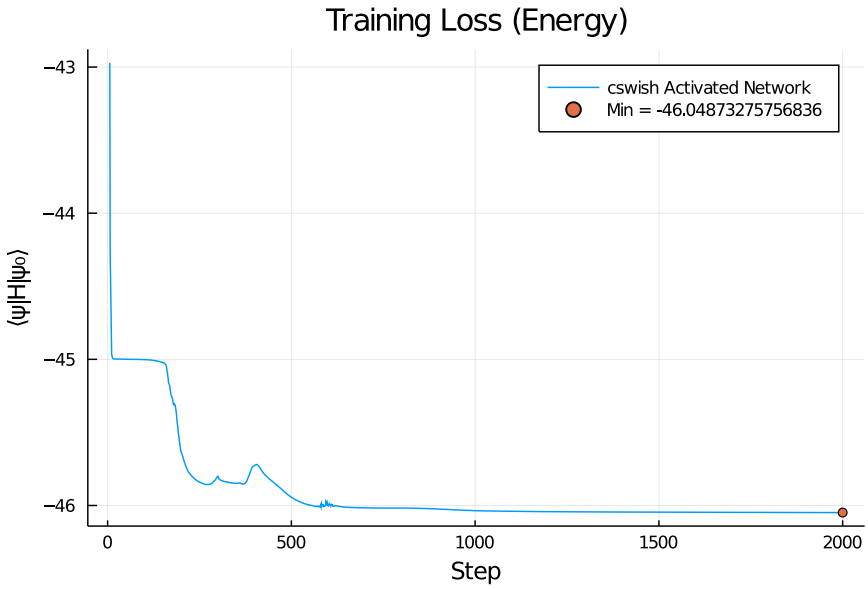

Figure A.8.: Training progress of the cswish activated network: energy vs step for the strong field model

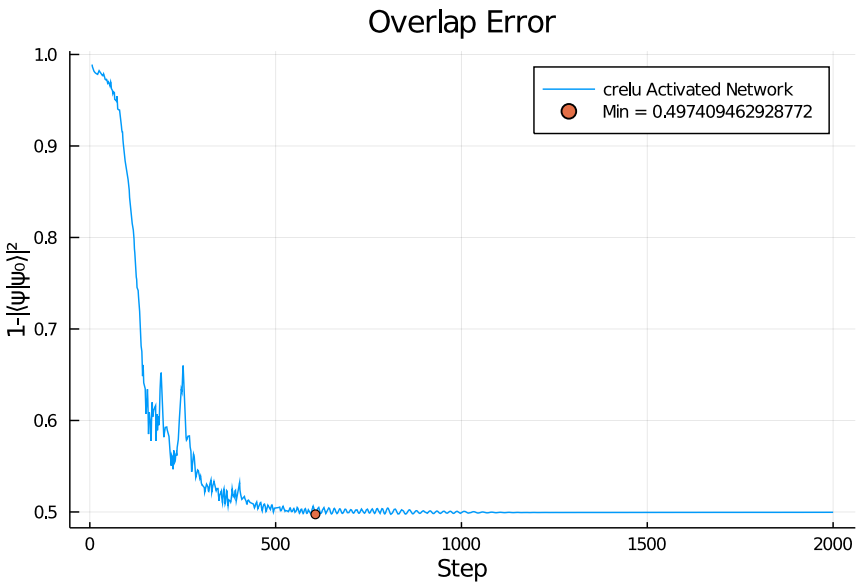

Figure A.9.: Training progress of the crelu activated network: overlap vs step for the weak field model

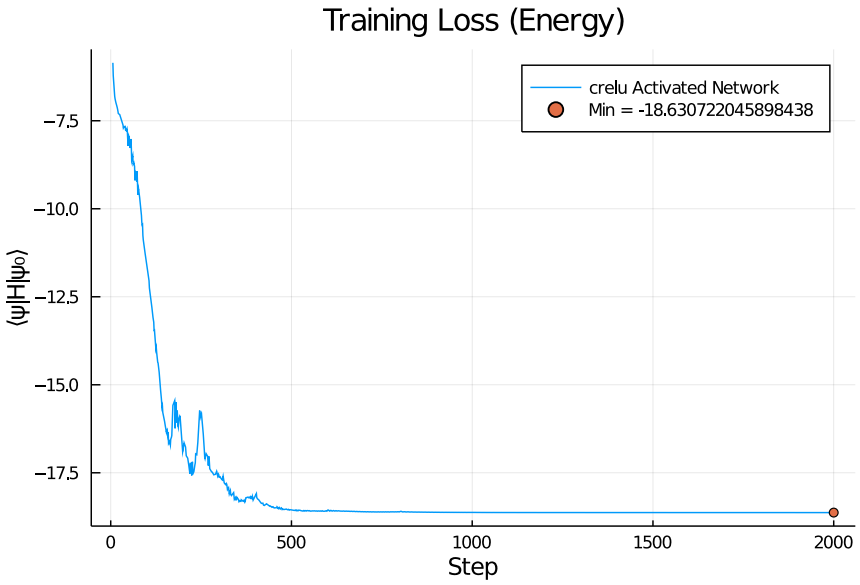

Figure A.10.: Training progress of the crelu activated network: energy vs step for the weak field model

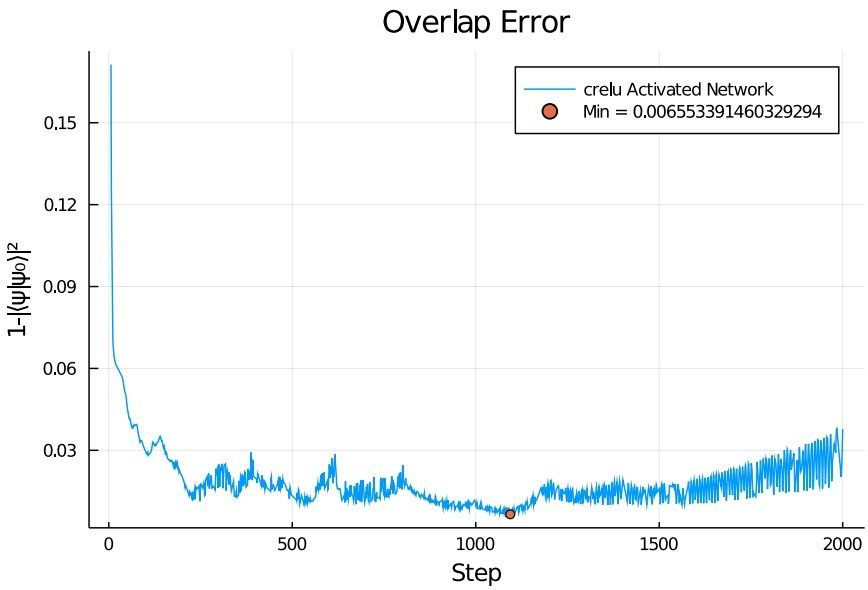

Figure A.11.: Training progress of the crelu activated network: overlap vs step for the strong field model

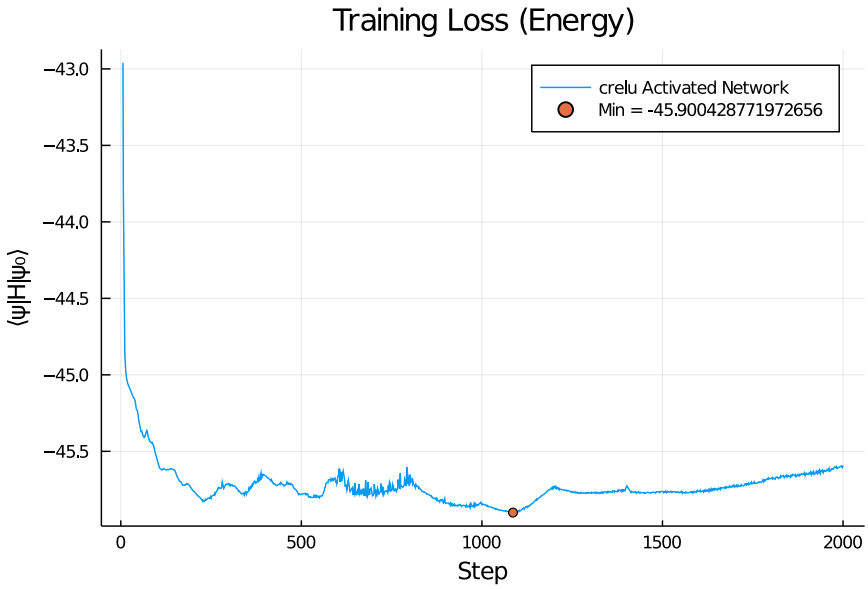

Figure A.12.: Training progress of the crelu activated network: energy vs step for the strong field model

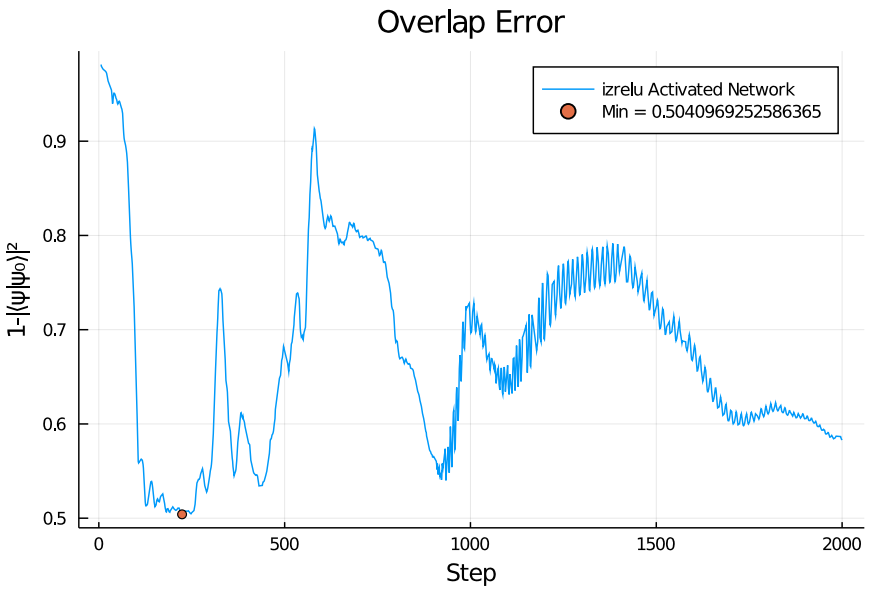

Figure A.13.: Training progress of the izrelu activated network: overlap vs step for the weak field model

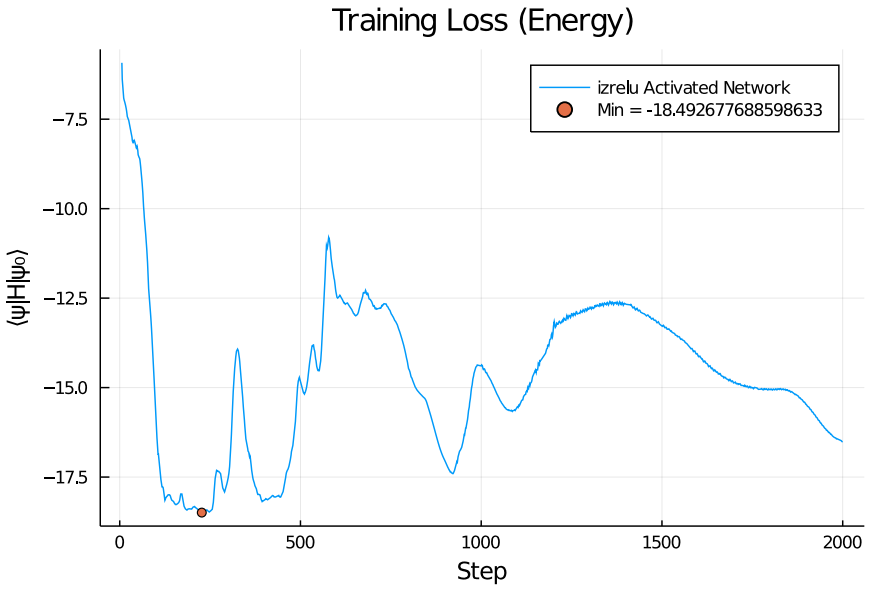

Figure A.14.: Training progress of the izrelu activated network: energy vs step for the weak field model

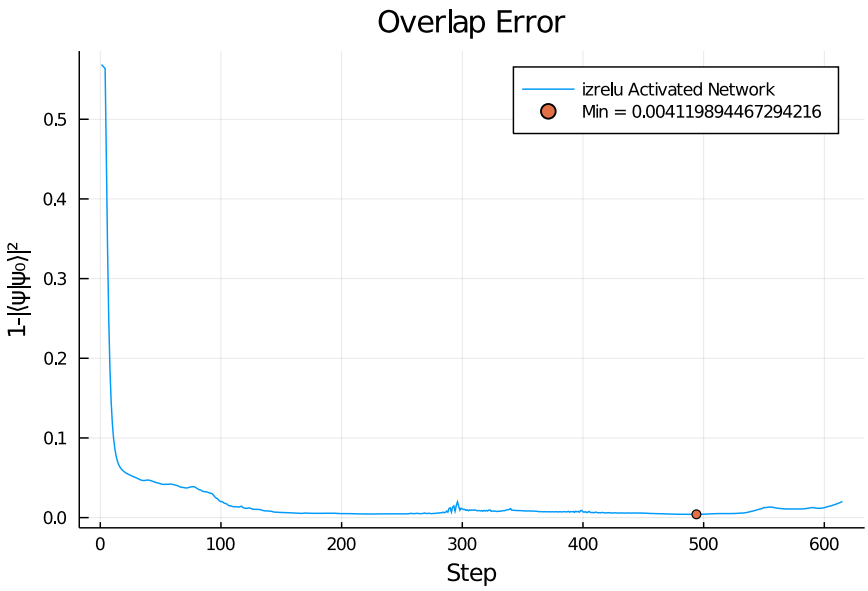

Figure A.15.: Training progress of the izrelu activated network: overlap vs step for the strong field model

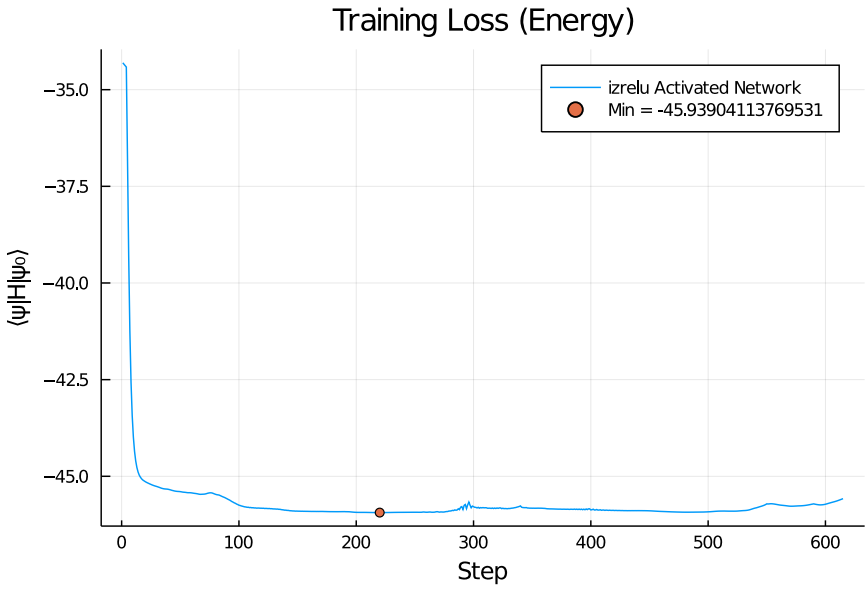

Figure A.16.: Training progress of the izrelu activated network: energy vs step for the strong field model

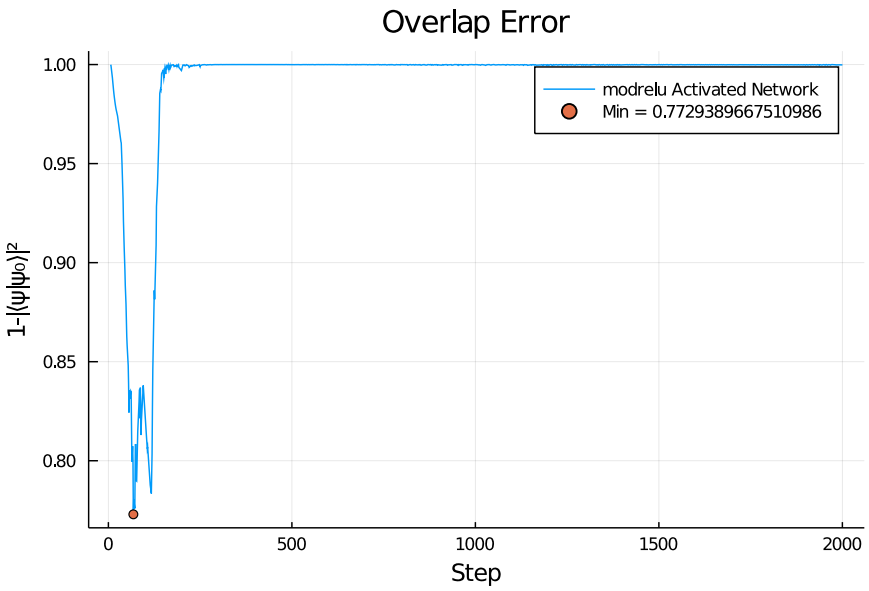

Figure A.17.: Training progress of the modrelu activated network: overlap vs step for the weak field model

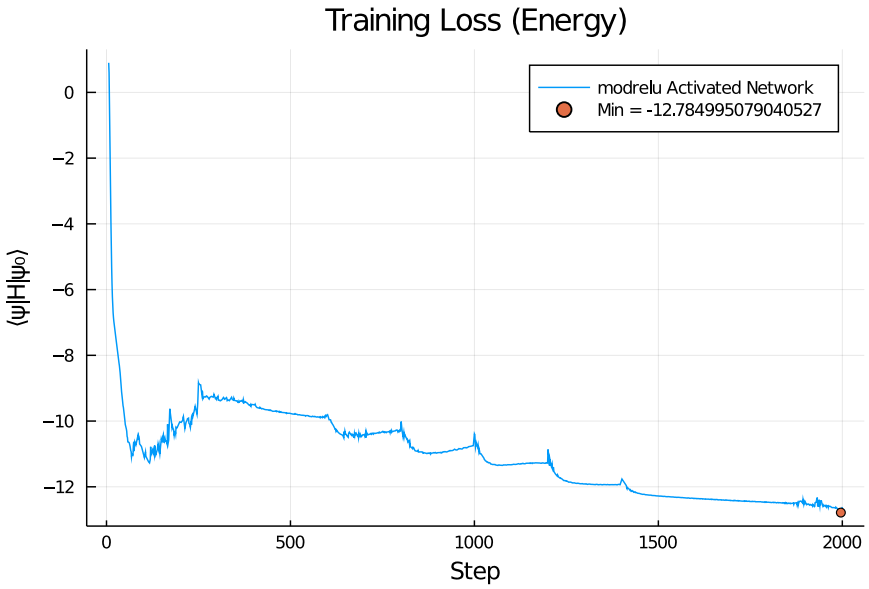

Figure A.18.: Training progress of the modrelu activated network: energy vs step for the weak field model

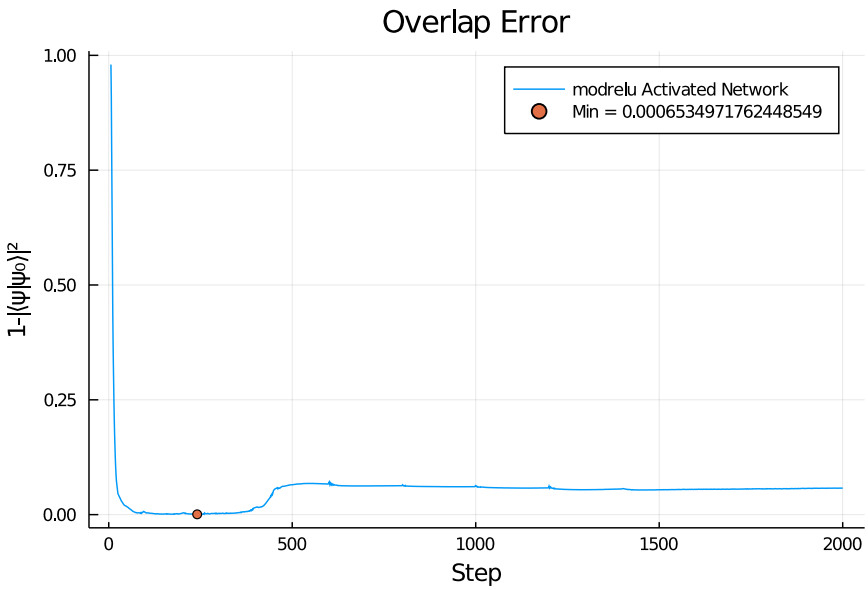

Figure A.19.: Training progress of the modrelu activated network: overlap vs step for the strong field model

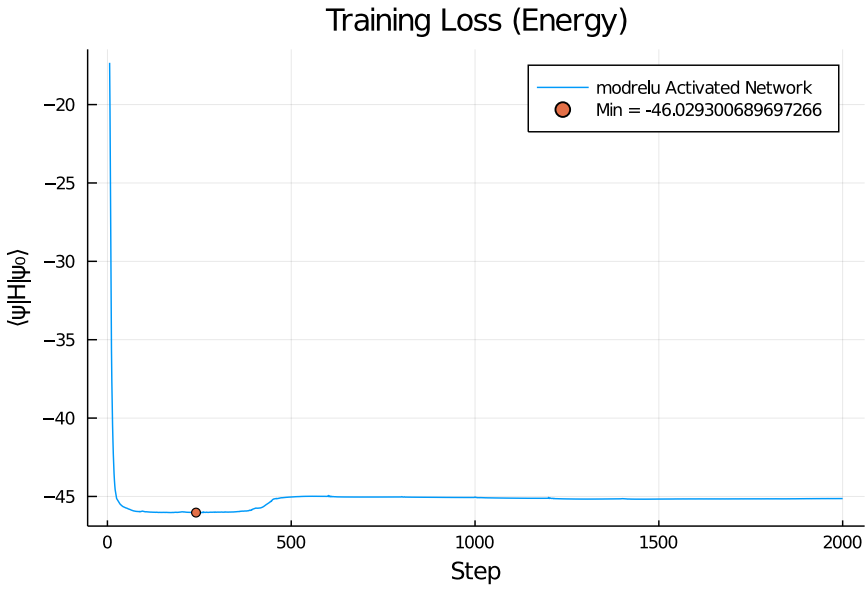

Figure A.20.: Training progress of the modrelu activated network: energy vs step for the strong field model

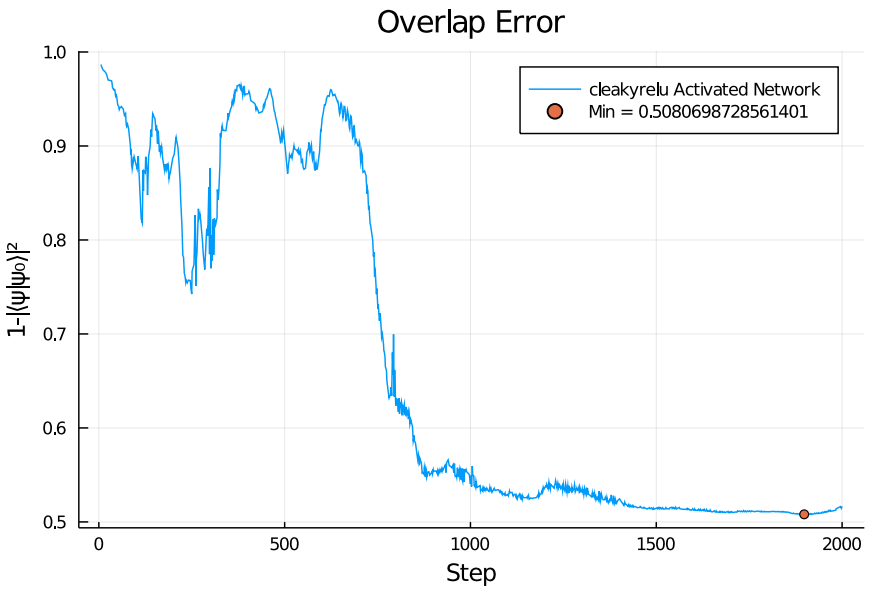

Figure A.21.: Training progress of the cleakyrelu activated network: overlap vs step for the weak field model

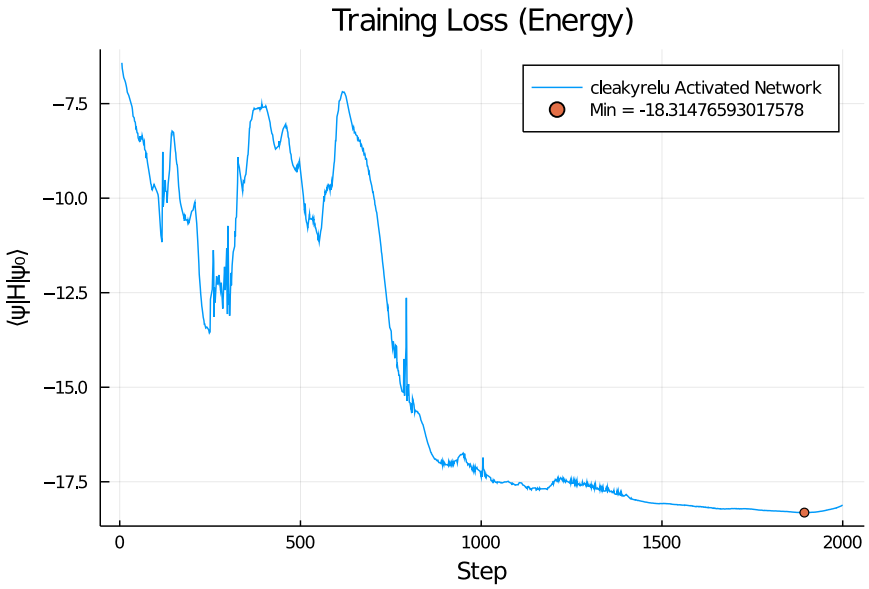

Figure A.22.: Training progress of the cleakyrelu activated network: energy vs step for the weak field model

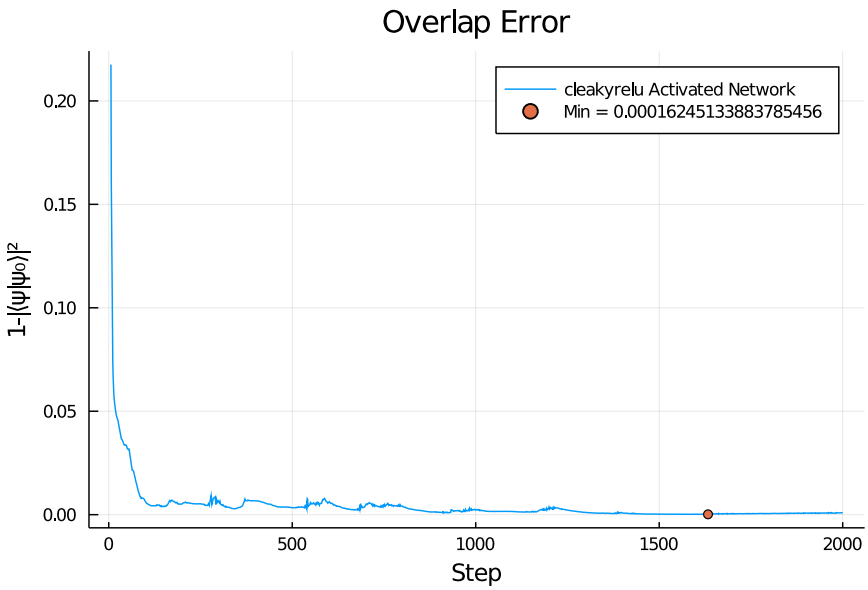

Figure A.23.: Training progress of the cleakyrelu activated network: overlap vs step for the strong field model

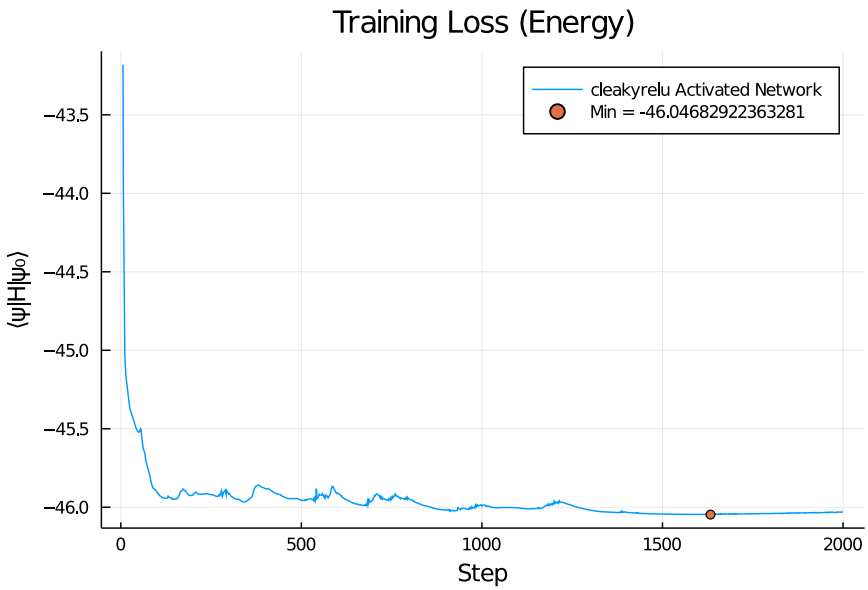

Figure A.24.: Training progress of the cleakyrelu activated network: energy vs step for the strong field model

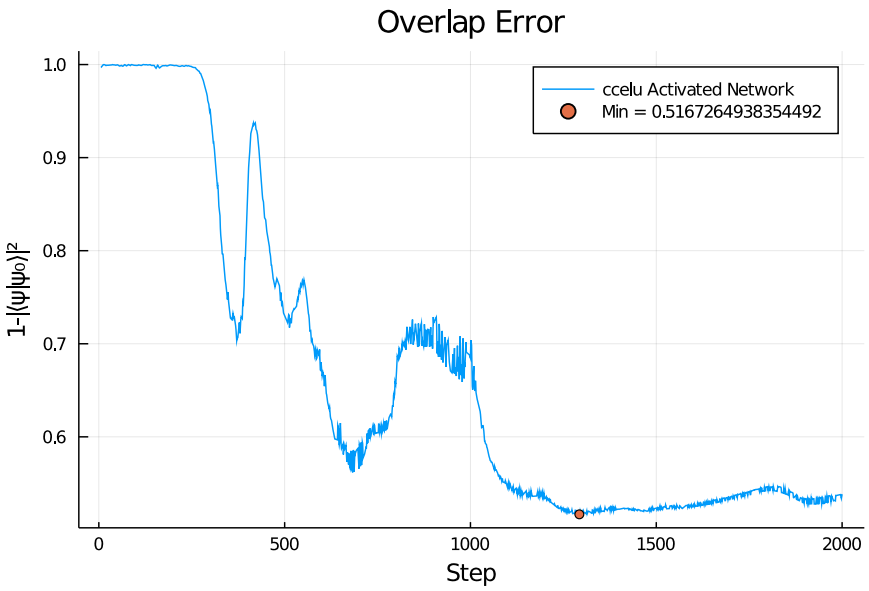

Figure A.25.: Training progress of the ccelu activated network: overlap vs step for the weak field model

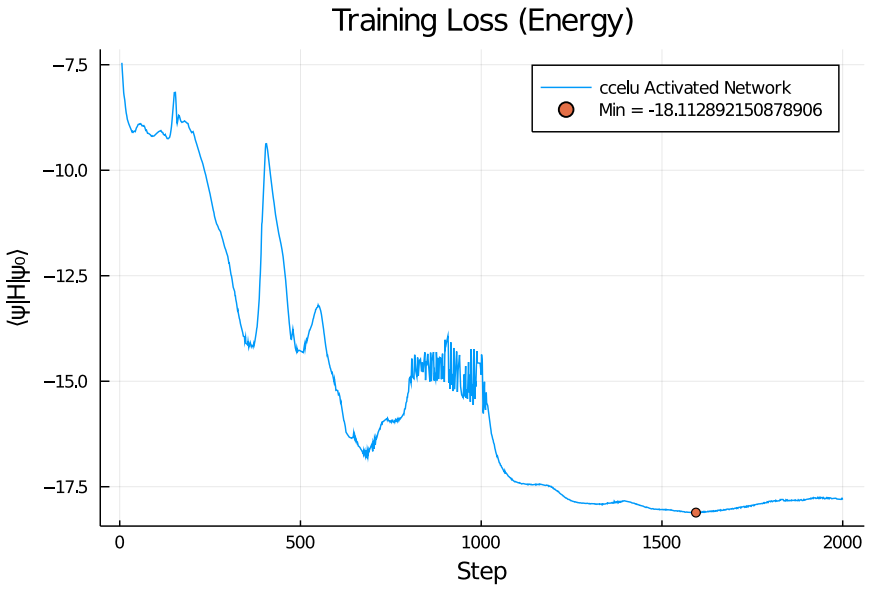

Figure A.26.: Training progress of the ccelu activated network: energy vs step for the weak field model

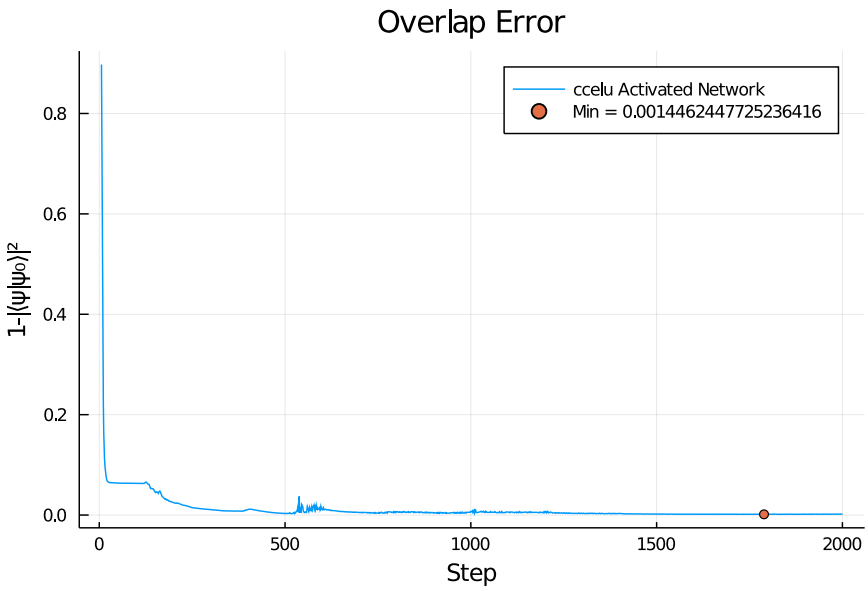

Figure A.27.: Training progress of the ccelu activated network: overlap vs step for the strong field model

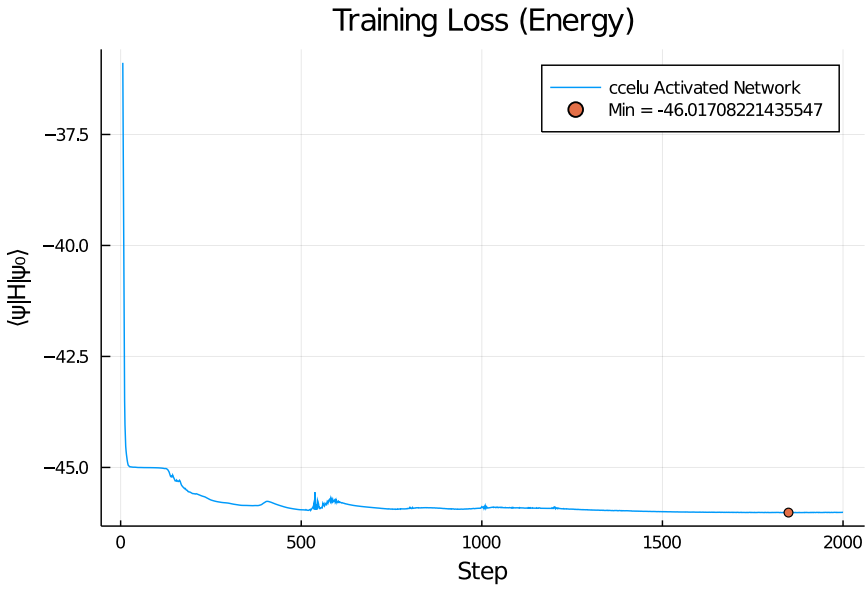

Figure A.28.: Training progress of the ccelu activated network: energy vs step for the strong field model

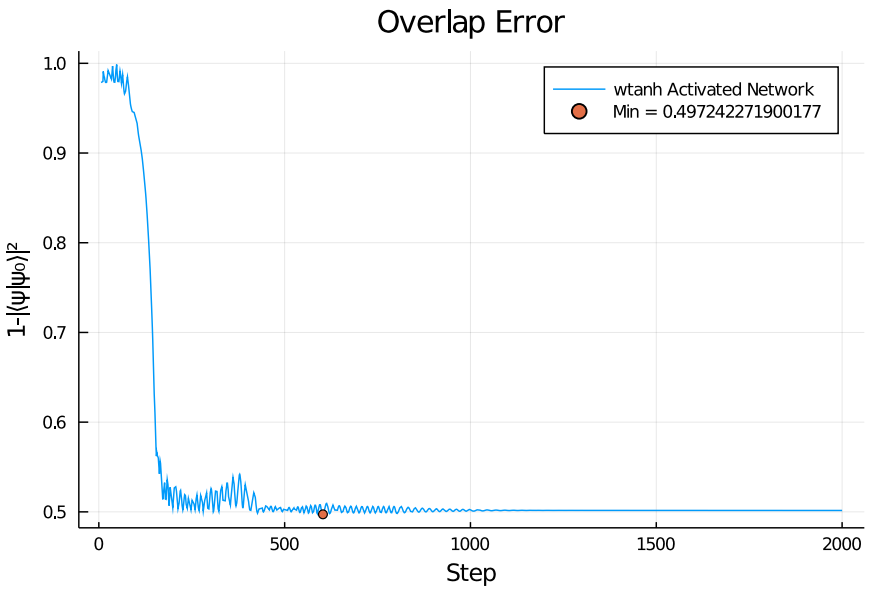

Figure A.29.: Training progress of the wtanh activated network: overlap vs step for the weak field model

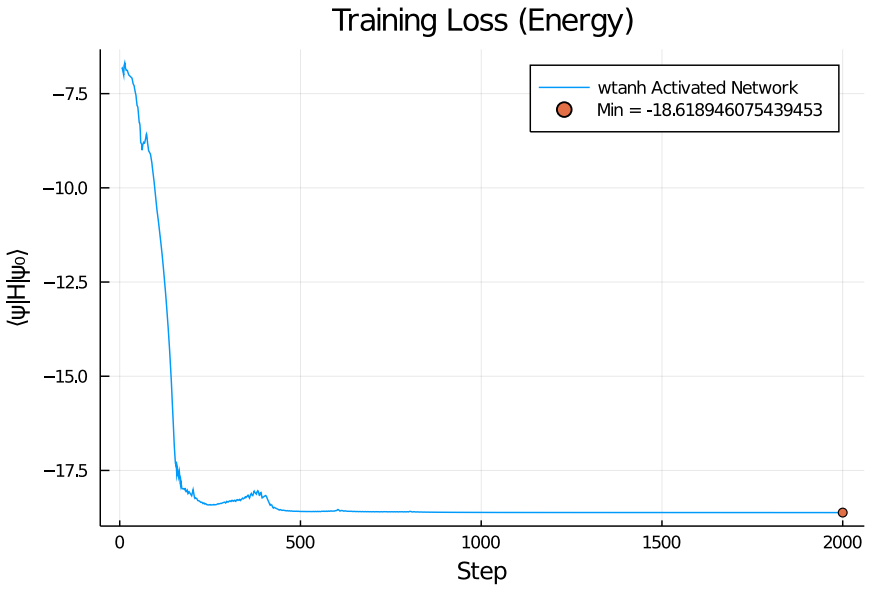

Figure A.30.: Training progress of the wtanh activated network: energy vs step for the weak field model

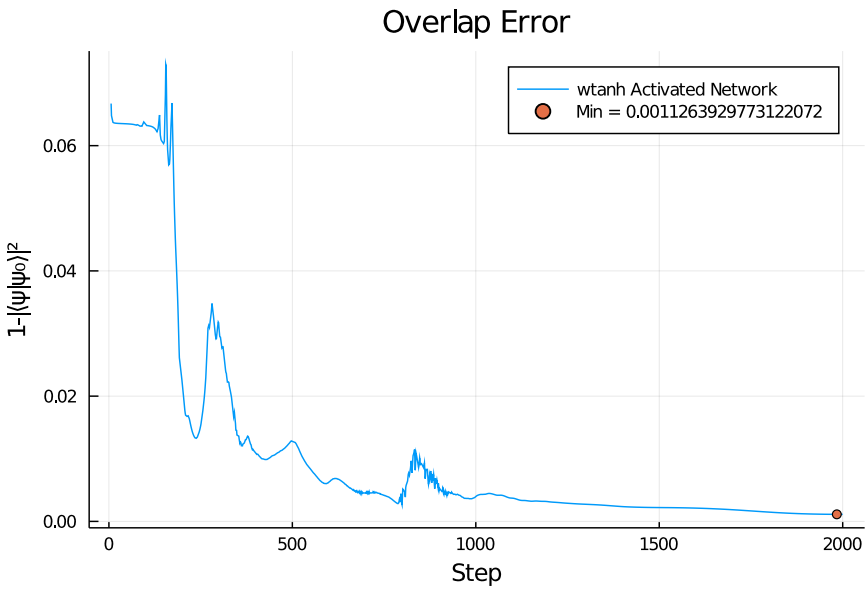

Figure A.31.: Training progress of the wtanh activated network: overlap vs step for the strong field model

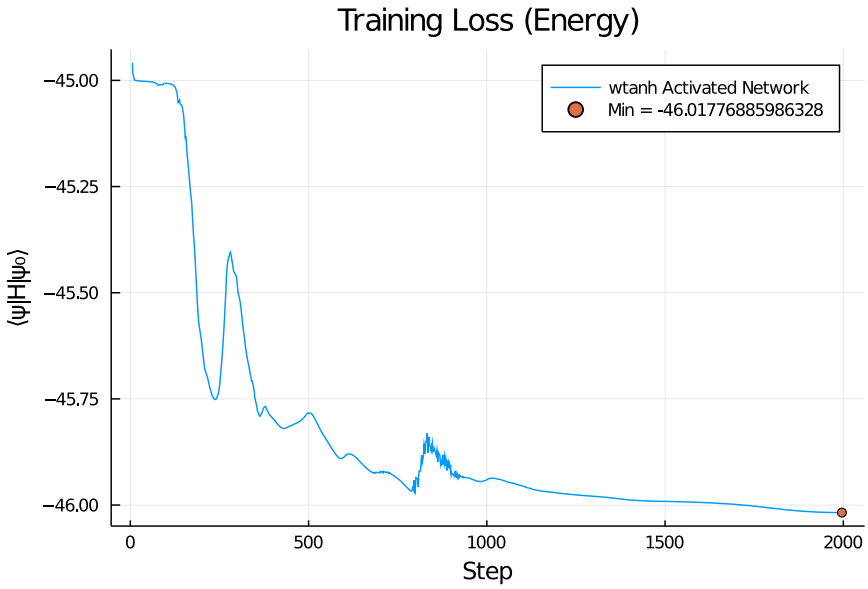

Figure A.32.: Training progress of the wtanh activated network: energy vs step for the strong field model

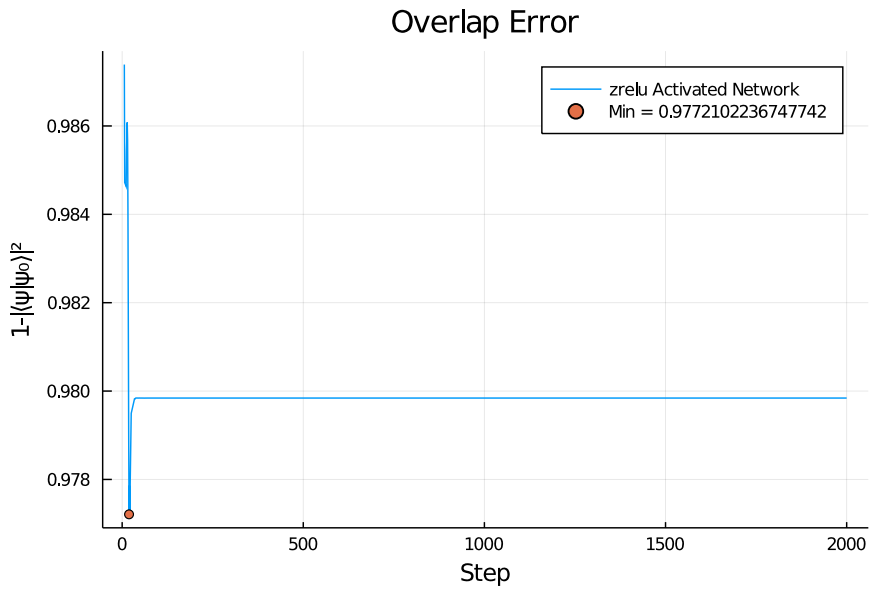

Figure A.33.: Training progress of the zrelu activated network: overlap vs step for the weak field model

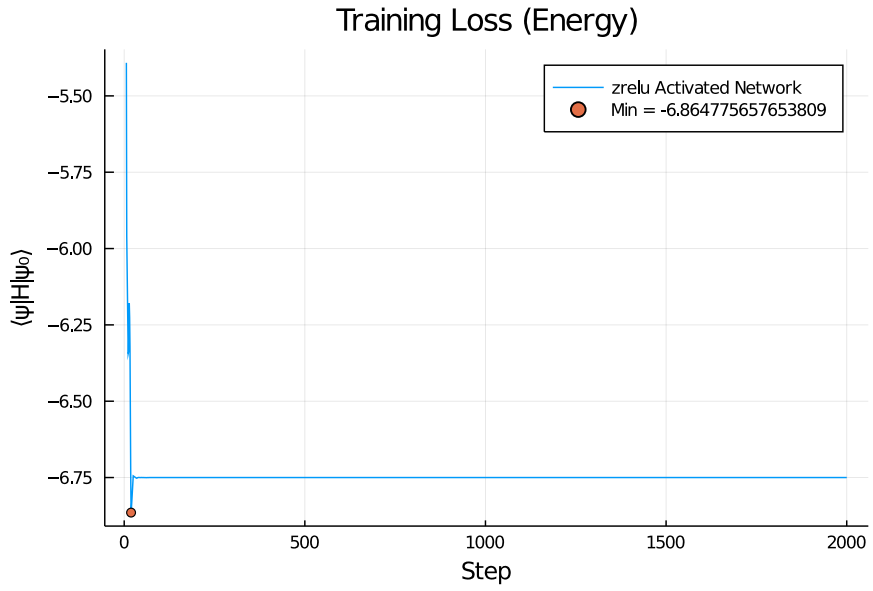

Figure A.34.: Training progress of the zrelu activated network: energy vs step for the weak field model

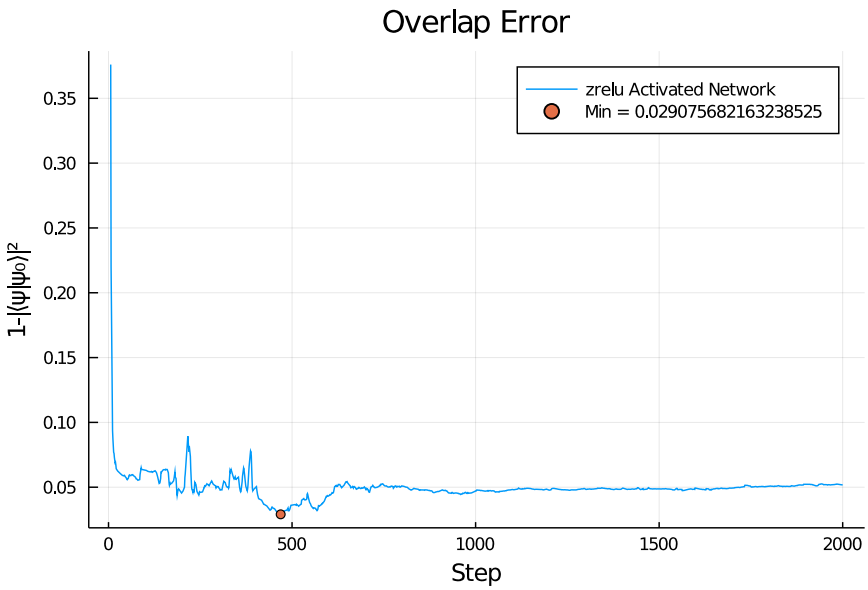

Figure A.35.: Training progress of the zrelu activated network: overlap vs step for the strong field model

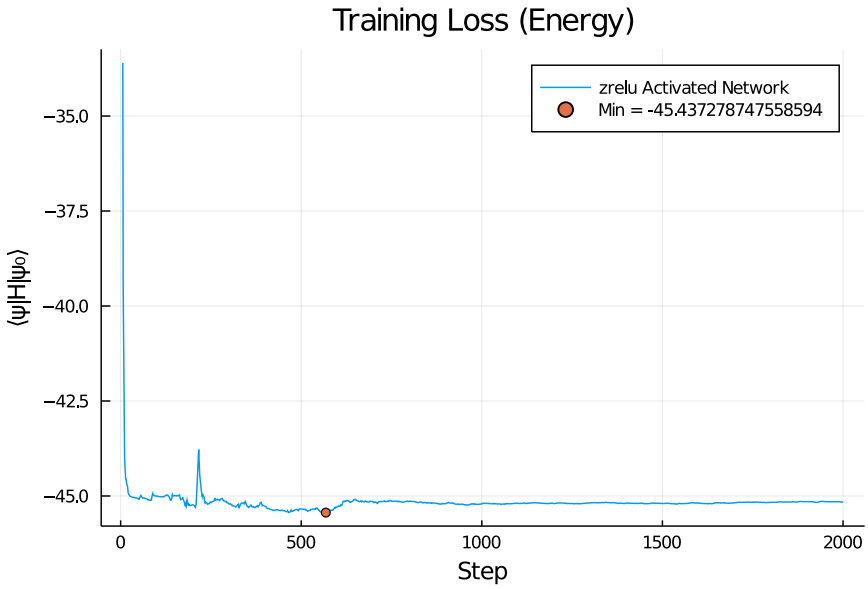

Figure A.36.: Training progress of the zrelu activated network: energy vs step for the strong field model

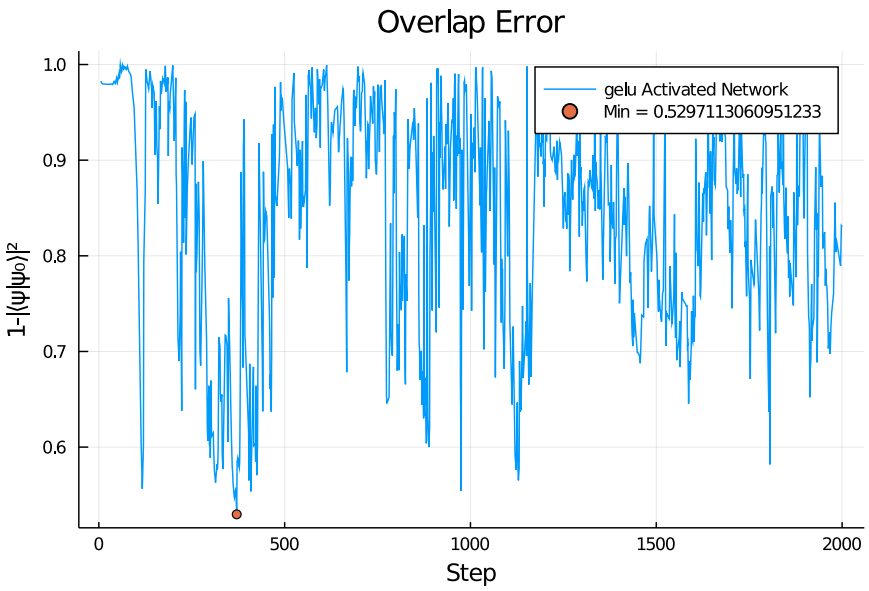

Figure A.37.: Training progress of the gelu activated network: overlap vs step for the weak field model

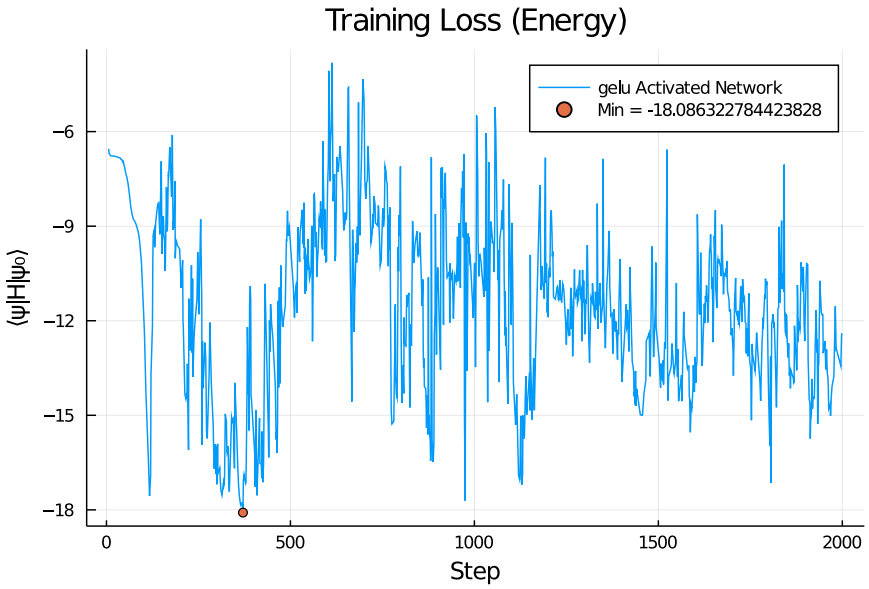

Figure A.38.: Training progress of the gelu activated network: energy vs step for the weak field model

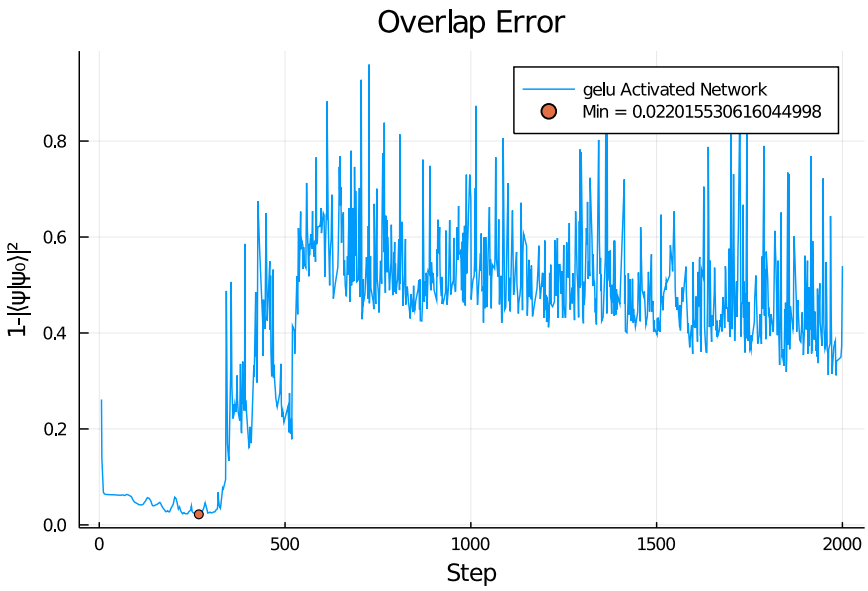

Figure A.39.: Training progress of the gelu activated network: overlap vs step for the strong field model

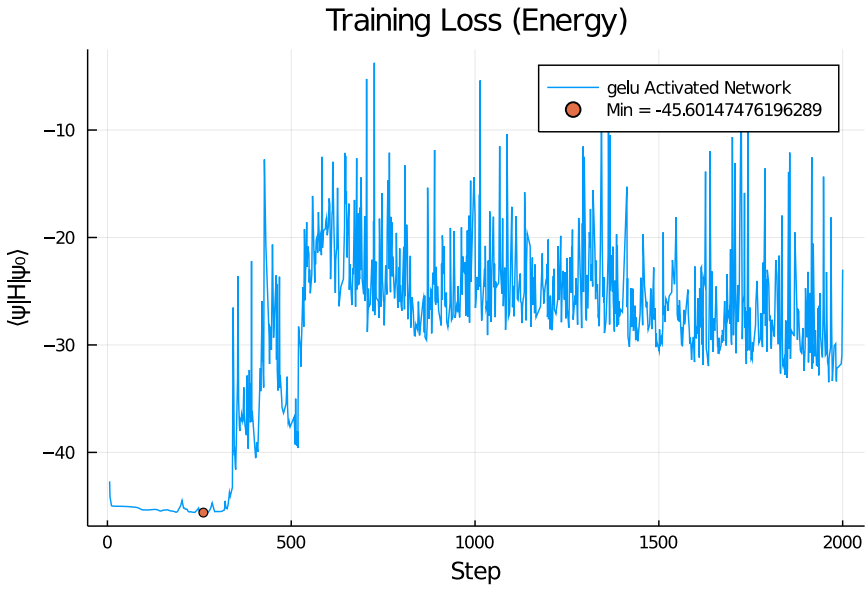

Figure A.40.: Training progress of the gelu activated network: energy vs step for the strong field model

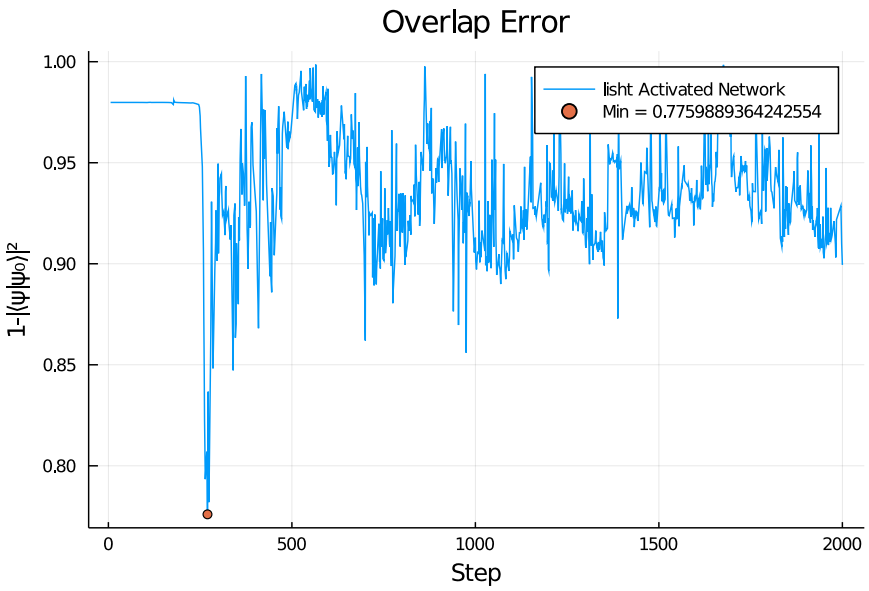

Figure A.41.: Training progress of the lisht activated network: overlap vs step for the weak field model

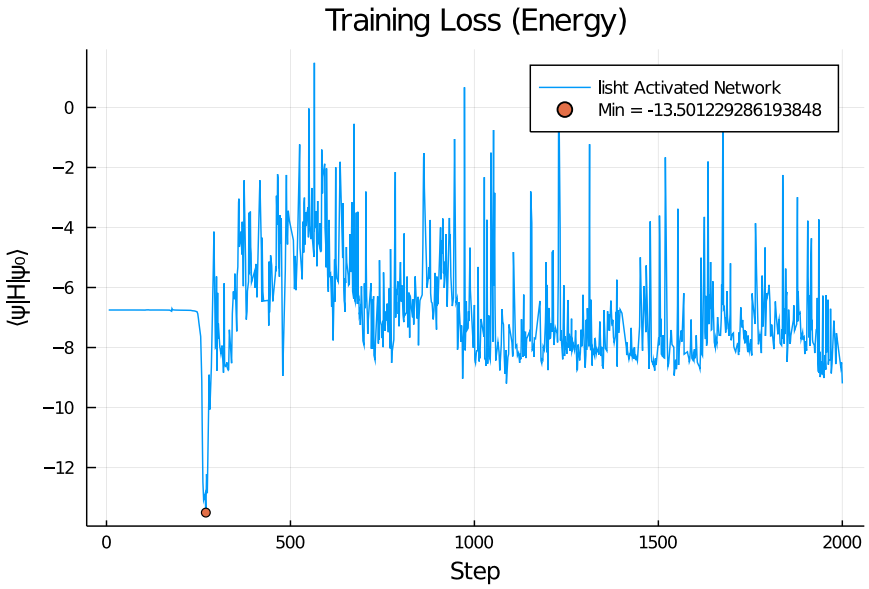

Figure A.42.: Training progress of the lisht activated network: energy vs step for the weak field model

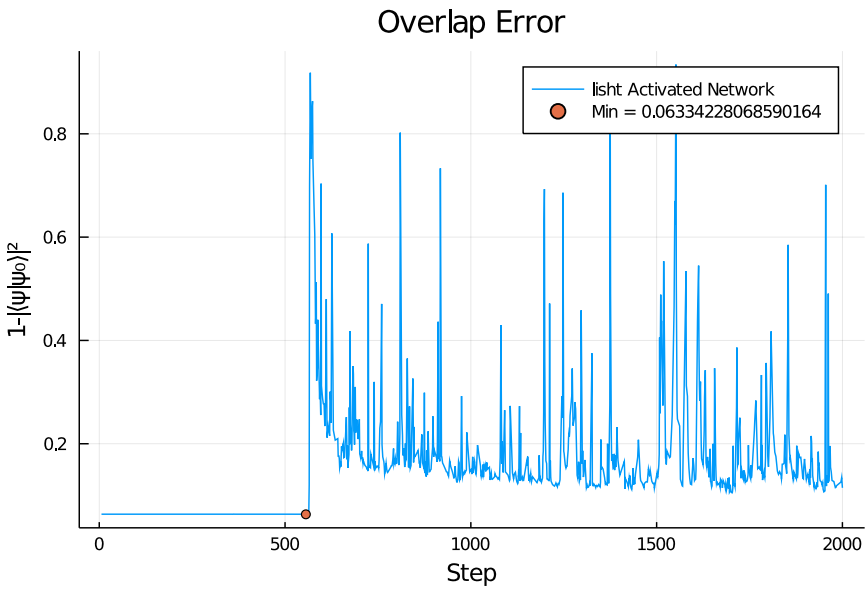

Figure A.43.: Training progress of the lisht activated network: overlap vs step for the strong field model

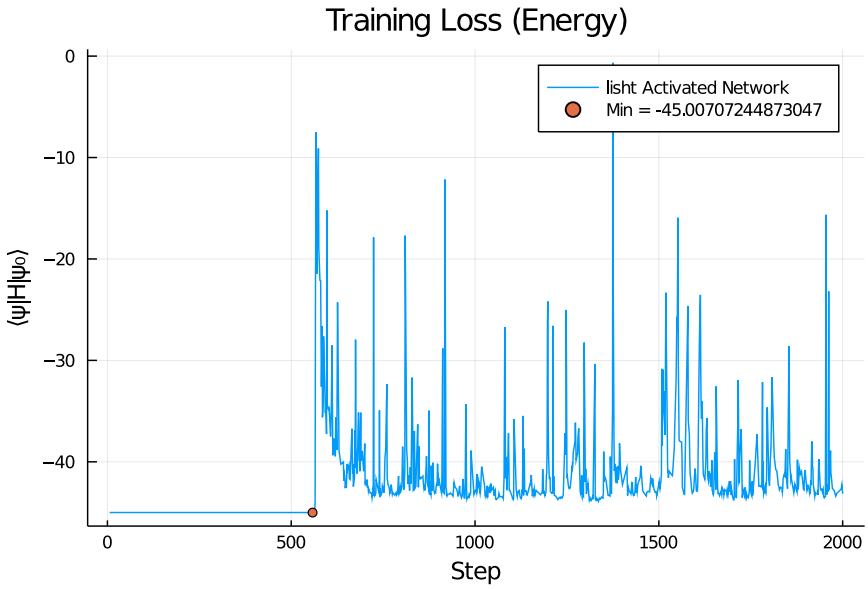

Figure A.44.: Training progress of the lisht activated network: energy vs step for the strong field model

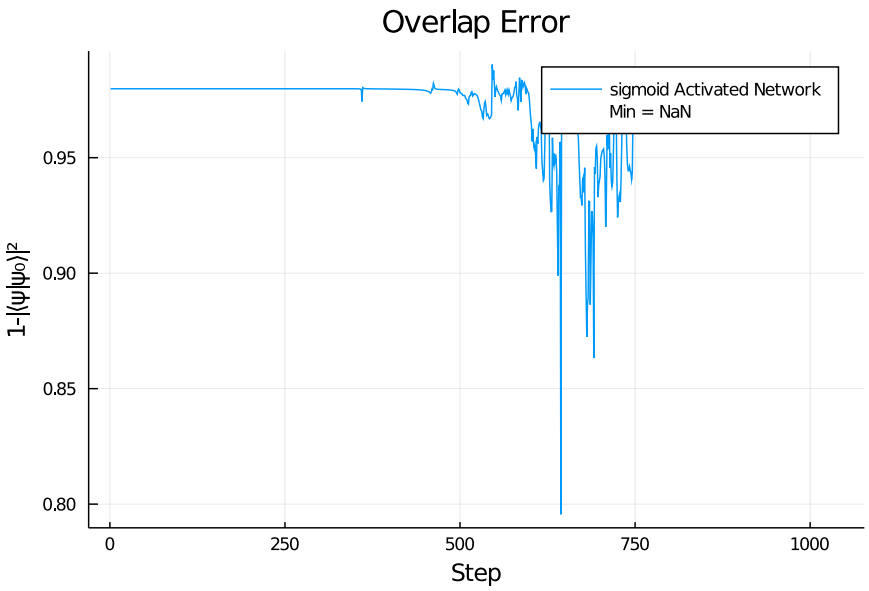

Figure A.45.: Training progress of the sigmoid activated network: overlap vs step for the weak field model

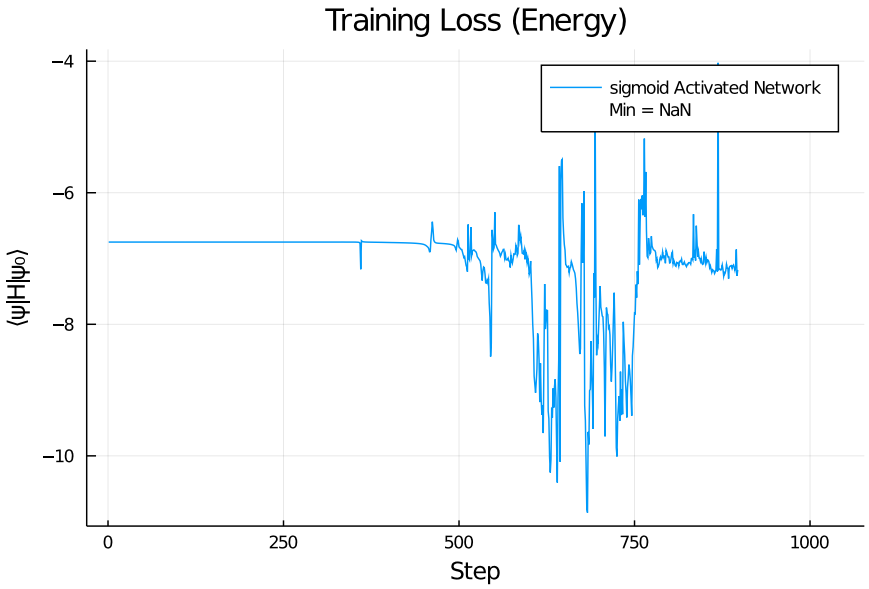

Figure A.46.: Training progress of the sigmoid activated network: energy vs step for the weak field model

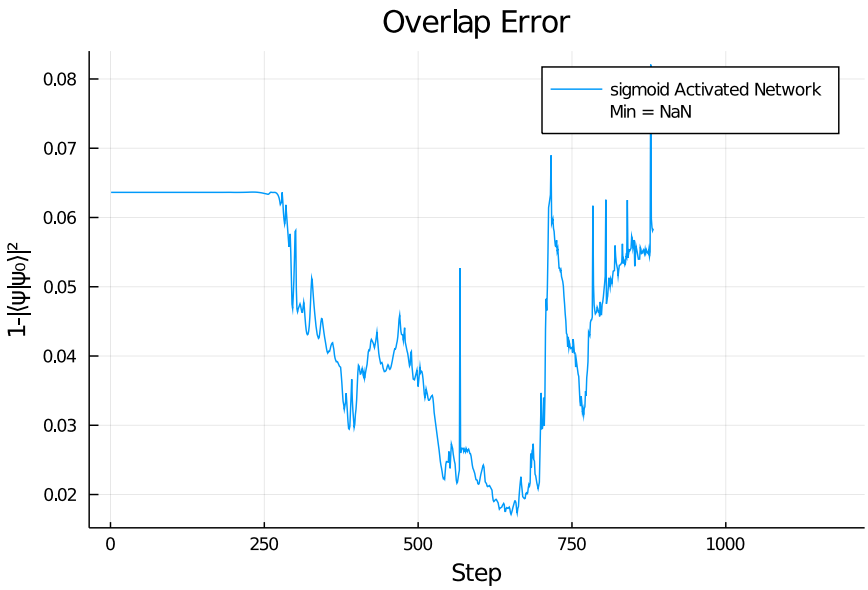

Figure A.47.: Training progress of the sigmoid activated network: overlap vs step for the strong field model

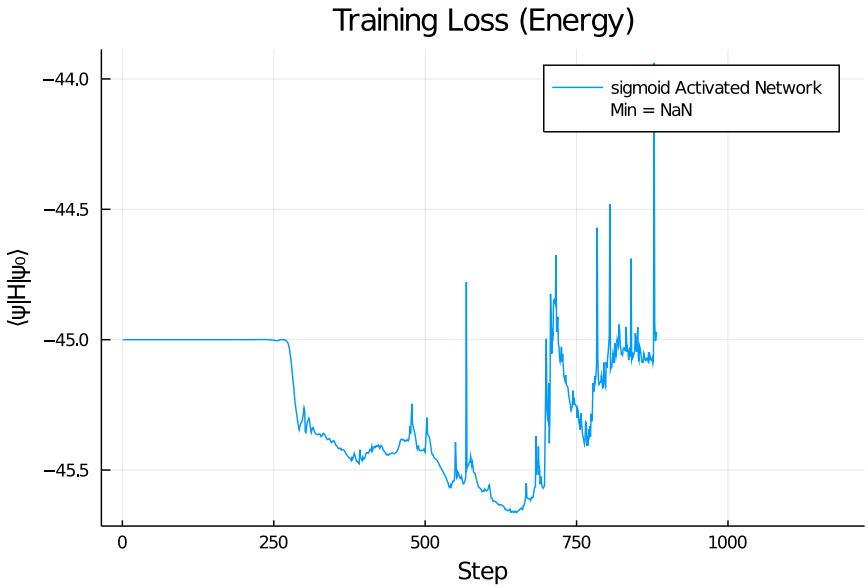

Figure A.48.: Training progress of the sigmoid activated network: energy vs step for the strong field model

B. T-NANODE Polynomials

Since we added the time dependent dimensions as learnable parameters to the optimizer, it might be helpful to visualize the polynomial functions that this network has acquired after optimization. The following figure shows values of the **Real** part of select complex parameters as time changes from $0 \rightarrow 10$.

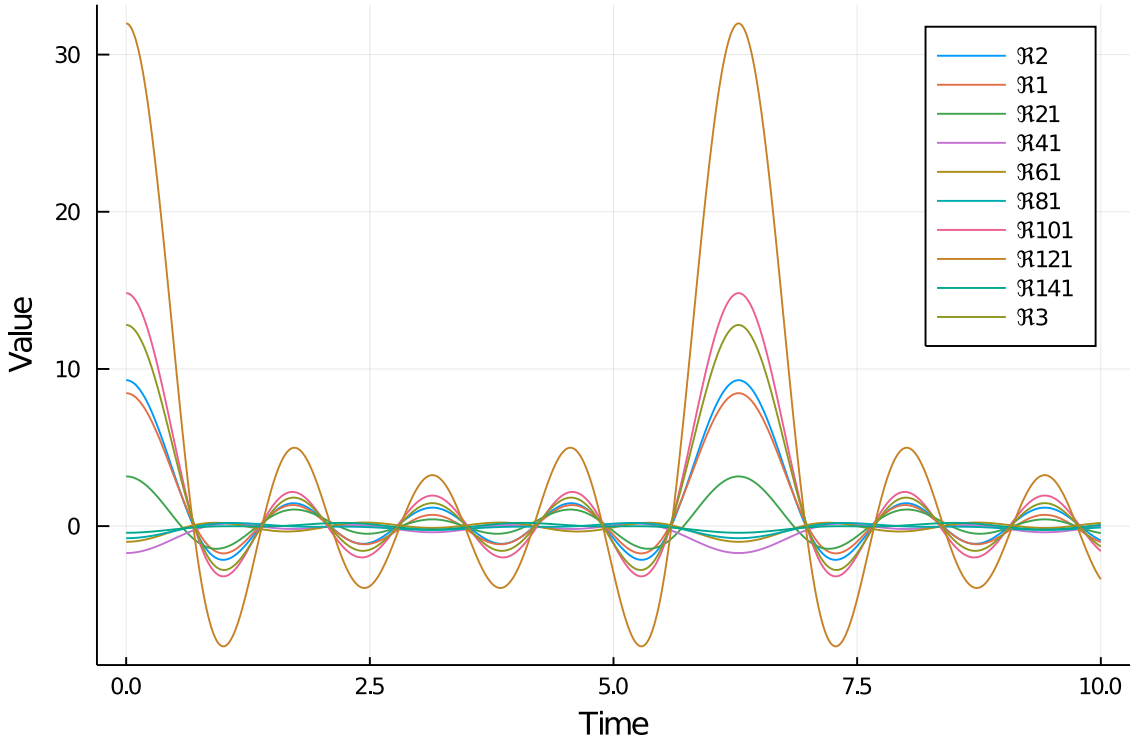

Figure B.1.: Parameter real value vs time for a sample of parameters

The following figure shows values of the **Imaginary** part of select complex parameters as time changes from $0 \rightarrow 10$.

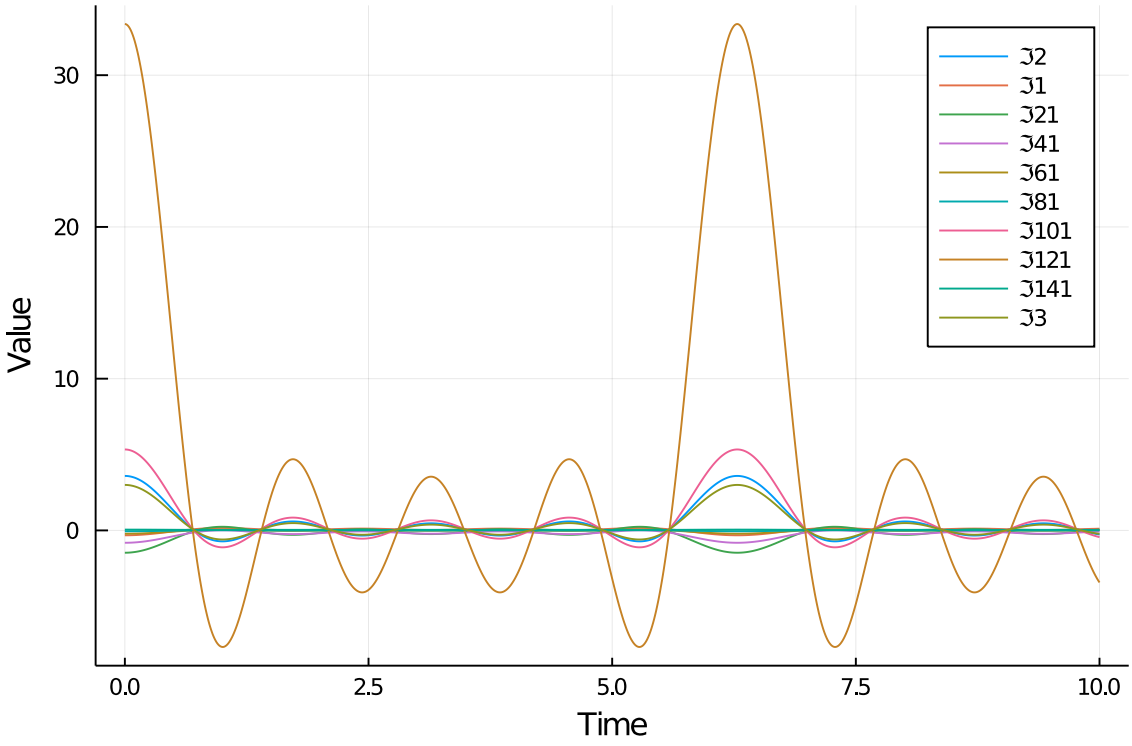

Figure B.2.: Parameter imaginary value vs time for a sample of parameters

Glossary

AdaBelief an adaptive optimizer that changes the stepsize according to the "belief" in the current gradient direction using the exponential moving average (EMA) of the noisy gradient.. 36

ADAM Adaptive Moment Estimation. 59

Adaptive Moment Estimation An algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments (see [49]). 59

ANODE Augmented Neural Ordinary Differential Equation. 14, 52

bound state A Bound State is a quantum state of a particle subject to a potential such that the particle has a tendency to remain localized in one or more regions of space. 4

CELU Continuously differentiable Exponential Linear Unit. 24, 25

CNN convolutional neural network. 29

ELU Exponential Linear Unit. 24, 25

ground state The lowest energy state of a quantum system. 3, 4, 31, 35, 36, 51, 52

NANODE Non-Autonomous Neural Ordinary Differential Equation. 51, 52

NODE Neural Ordinary Differential Equation. 12, 14, 30, 45, 51, 52

ODE Ordinary Differential Equation. 52

RADAM Rectified ADAM. 35, 36

RBM Restricted Bolzmann Machine. 52

Rectified ADAM a variant of Adaptive Moment Estimation (ADAM), characterized by introducing a term to rectify the variance of the adaptive learning rate. (see [50]). 35, 59

RELU **Re**ctified **L**inear **U**nit. 13, 17

same padding A padding scheme that will result in an output that has the same size as the input. 30

TFIM Transverse Field Ising Model. 7, 10, 30, 36, 39, 49, 51

T-NANODE Trigonometric (Polynomials) Non-Autonomous Neural Ordinary Differential Equation. 32, 33, 36, 45, 49, 51, 52

Bibliography

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

[2] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, Jun 1982. ISSN 1572-9575. doi: 10.1007/BF02650179. URL https://doi.org/10.1007/BF02650179.

[3] David Poulin, Angie Qarry, Rolando Somma, and Frank Verstraete. Quantum simulation of time-dependent hamiltonians and the convenient illusion of hilbert space. *Phys. Rev. Lett.*, 106:170501, Apr 2011. doi: 10.1103/PhysRevLett.106.170501. URL https://link.aps.org/doi/10.1103/PhysRevLett.106.170501.

[4] R B Stinchcombe. Ising model in a transverse field. i. basic theory. *Journal of Physics C: Solid State Physics*, 6(15):2459–2483, aug 1973. doi: 10.1088/0022-3719/6/15/009. URL https://doi.org/10.1088%2F0022-3719%2F6%2F15%2F009.

[5] P.G. de Gennes. Collective motions of hydrogen bonds. *Solid State Communications*, 1(6):132 – 137, 1963. ISSN 0038-1098. doi: https://doi.org/10.1016/0038-1098(63)90212-6. URL http://www.sciencedirect.com/science/article/pii/0038109863902126.

[6] Yasusada Yamada and Takemi Yamada. Inter-dipolar interaction in nano2. *Journal of the Physical Society of Japan*, 21(11):2167–2177, 1966. doi: 10.1143/JPSJ.21.2167. URL https://doi.org/10.1143/JPSJ.21.2167.

[7] Dan Zhang and Anna Khoreva. Progressive augmentation of gans, 2019.

[8] Andrew Tao, Karan Sapra, and Bryan Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*, 2020.

[9] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *arXiv preprint arXiv:1611.04717*, 2016.

[10] Antonin Raffin and Freek Stulp. Generalized state-dependent exploration for deep reinforcement learning in robotics. *arXiv preprint arXiv:2005.05719*, 2020.

[11] Samuel S Schoenholz, Ekin D Cubuk, Daniel M Sussman, Efthimios Kaxiras, and Andrea J Liu. A structural approach to relaxation in glassy liquids. *Nature Physics*, 12 (5):469–471, 2016.

[12] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

[13] David J Griffiths and Darrell F Schroeter. *Introduction to quantum mechanics*. Cambridge University Press, 2018.

[14] Jim Branson Nikos Drakos, Ross Moore. University of california san diego: Lecture notes on quantum physics, 2003. URL https://quantummechanics.ucsd.edu/ph130a/130_notes/130_notes.html.

[15] C Figueira de Morisson Faria and A Fring. Time evolution of non-hermitian hamiltonian systems. *Journal of Physics A: Mathematical and General*, 39(29):9269–9289, jul 2006. doi: 10.1088/0305-4470/39/29/018. URL https://doi.org/10.1088/0305-4470/39/29/018.

[16] Carl M Bender, Dorje C Brody, and Hugh F Jones. Must a hamiltonian be hermitian? *American Journal of Physics*, 71(11):1095–1102, 2003.

[17] Carl M Bender and Philip D Mannheim. Exactly solvable p t-symmetric hamiltonian having no hermitian counterpart. *Physical Review D*, 78(2):025022, 2008.

[18] Wenxuan Huang, Daniil A Kitchaev, Stephen T Dacek, Ziqin Rong, Alexander Urban, Shan Cao, Chuan Luo, and Gerbrand Ceder. Finding and proving the exact ground state of a generalized ising model by convex optimization and max-sat. *Physical Review B*, 94(13):134424, 2016.

[19] Victor Y. Pan and Zhao Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, page 507–516, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130678. doi: 10.1145/301250.301389. URL https://doi.org/10.1145/301250.301389.

[20] Benedikt Fauseweh. Analysis of the transverse field ising model by continuous unitary transformations. *Technische Universität Dortmund*, 2012.

[21] Nimish Mishra, Manik Kapil, Hemant Rakesh, Amit Anand, Nilima Mishra, Aakash Warke, Soumya Sarkar, Sanchayan Dutta, Sabhyata Gupta, Aditya Prasad Dash, Rakshit Gharat, Yagnik Chatterjee, Shuvarati Roy, Shivam Raj, Valay Kumar Jain, Shreeram Bagaria, Smit Chaudhary, Vishwanath Singh, Rituparna Maji, Priyanka Dalei, Bikash K. Behera, Sabyasachi Mukhopadhyay, and Prasanta K. Panigrahi. Quantum machine learning: A review and current status. In Neha Sharma, Amlan Chakrabarti,

Valentina Emilia Balas, and Jan Martinovic, editors, *Data Management, Analytics and Innovation*, pages 101–145, Singapore, 2021. Springer Singapore. ISBN 978-981-15-5619-7.

[22] Eduardo Paluzo-Hidalgo, Rocio Gonzalez-Diaz, and Miguel A. Gutiérrez-Naranjo. Two-hidden-layer feed-forward networks are universal approximators: A constructive approach. *Neural Networks*, 131:29 – 36, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.07.021. URL http://www.sciencedirect.com/science/article/pii/S0893608020302628.

[23] D. E. Rumelhart and J. L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[24] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[25] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671.

[26] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018. doi: 10.21105/joss.00602.

[27] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018. URL https://arxiv.org/abs/1811.01457.

[28] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl - a julia library for neural differential equations, 2019.

[29] B. Speelpenning. Compiling fast partial derivatives of functions given by algorithms. *U.S. Department of Energy, Office of Scientific and Technical Information*, 1 1980. doi: 10.2172/5254402. URL https://www.osti.gov/biblio/5254402.

[30] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf.

[31] Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14(1):115–133, 1994.

[32] Grzegorz Lewicki and G Marino. Approximation by superpositions of a sigmoidal function. *Zeitschrift für Analysis und ihre Anwendungen*, 22(2):463–470, 2003.

[33] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[34] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[35] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

[36] Jared Quincy Davis, Krzysztof Choromanski, Jake Varley, Honglak Lee, Jean-Jacques Slotine, Valerii Likhosterov, Adrian Weller, Ameesh Makadia, and Vikas Sindhwani. Time dependence in non-autonomous neural odes, 2020.

[37] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[38] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes, 2019.

[39] Chiheb Trabelsi, Olexa Bilaniuk, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J. Pal. Deep complex networks. *CoRR*, abs/1705.09792, 2017. URL http://arxiv.org/abs/1705.09792.

[40] Nitzan Guberman. On complex valued convolutional neural networks. *CoRR*, abs/1602.09046, 2016. URL http://arxiv.org/abs/1602.09046.

[41] Martín Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *CoRR*, abs/1511.06464, 2015. URL http://arxiv.org/abs/1511.06464.

[42] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[43] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[44] Jonathan T. Barron. Continuously differentiable exponential linear units. *CoRR*, abs/1704.07483, 2017. URL http://arxiv.org/abs/1704.07483.

[45] Stefanie Czischek, Martin Gärttner, and Thomas Gasenzer. Quenches near ising quantum criticality as a challenge for artificial neural networks. *Physical Review B*, 98(2), Jul 2018. ISSN 2469-9969. doi: 10.1103/physrevb.98.024311. URL http://dx.doi.org/10.1103/PhysRevB.98.024311.

[46] Irene L'opez-Guti'errez and Christian B. Mendl. Real time evolution with neural-network quantum states. *arXiv: Disordered Systems and Neural Networks*, 2019.

[47] Swalpa Kumar Roy, Suvojit Manna, Shiv Ram Dubey, and Bidyut Baran Chaudhuri. Lisht: Non-parametric linearly scaled hyperbolic tangent activation function for neural networks, 2020.

[48] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2020.

[49] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[50] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *CoRR*, abs/1908.03265, 2019. URL http://arxiv.org/abs/1908.03265.