Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Daniel Richter

born in Miltenberg (Germany)

2022

Laser pulse reconstruction in transient absorption spectroscopy using machine-learning regression models

This Bachelor Thesis has been carried out by Daniel Richter at the
Max Planck Institute for Nuclearphysics in Heidelberg
under the supervision of
Prof. Dr. Thomas Pfeifer

Abstract

This thesis sets up a machine learning based numerical approach to reconstructing properties of ultrafast laser pulses which are utilized in absorption spectroscopy. A comparison of different machine learning models applied on transient absorption spectroscopy for different noise levels will be presented.
More specifically, in the framework of a numerical few-level system, absorption spectra are generated by randomly sampling the pulse parameters of the incident laser pulse to generate data sets for the machine learning algorithms to train on. Additionally, different levels of gaussian noise are applied to the spectra to mimic measurement data. These datasets are used to train, test and evaluate various machine learning models, ranging from the most fundamental models to state of the art models developed within the last twenty years.
The trained models are then used to reconstruct the parameters of the generating laser pulse from individual absorption spectra. The performance of different models for this task is compared and the influence of noisy data on the different models is investigated.

Zusammenfassung

In dieser Arbeit geht es um einen Vergleich verschiedener Arten von maschinellen Lernmodellen, die auf die transiente Absorptionsspektroskopie angewendet werden. Im Rahmen eines numerischen few-level Systems werden Absorptionsspektren durch zufällige Variation der Pulsparameter des einfallenden Laserpulses erzeugt. Zusätzlich werden die Spektren mit unterschiedlichen Stufen von Gaußschem Rauschen versehen, um experimentelle Daten zu imitieren. Anhand dieser Datensätze werden verschiedene Modelle mit maschinellem Lernen trainiert, getestet und evaluiert. Die verwendeten Modelle reichen dabei von den grundlegendsten Modellen, wie z.B. Entscheidungsbäumen, bis hin zu, in den letzten zwanzig Jahren entwickelten modernsten Modellen, wie beispielsweise Neuronale Faltungsnetwerke.
Die trainierten Modelle werden dann verwendet, um die Pulsparameter des einfallenden Laserlichts aus einzelnen Absorptionsspektren zu rekonstruieren. Die Performance verschiedener Modelle für diese Aufgabe wird verglichen und der Einfluss von Rauschen auf den Daten auf die verschiedenen Modelle untersucht.

Contents

1 Introduction

Today, our deepest understanding of the atomic world stems from the advent of quantum mechanics. This theory made it is possible to describe the physical properties of nature at the scale of atoms and molecules.
To be able to gain further knowledge about the dynamical behavior of these systems, special measurement techniques are needed. Dynamics at the atomic scale evolve at rapid speeds, thus an equally fast measurement method is needed. Ultrashort laser pulses in the range of femtoseconds have proven to be a viable option to resolve these ultafast procesess,e.g. electronic processes in atoms. Transient absorption spectroscopy measures changes in the transmittance of a sample medium by comparing incoming intensities of ultrashort light pulses with the transmitted light. Calculating the optical density has proven to be a valuable tool to interpret and extract knowledge about the dynamical behavior of systems. However, this, mainly analytical solution is limited to weak field interactions, while for stronger fieldstrenghts conventional analytical models no longer apply.
An emerging field of study aimed at solving the kind of problems traditional algorithms struggle with, is so-called Machine Learning (ML). First conceptualized in the 1940's [1], it began with the development of artificial neurons modeled closely after biological neurons. Only in recent decades, with the rise of the modern computer, large scale network based learning models were able to reach human level or in recent years, even surpass human level performance in a variety of tasks. Today it is possible to propose, build and train artificial neural networks to solve many different tasks requiring learning-based solutions. The speed, memory size and performance required to effectively train semi-large neural networks has been reached, even by consumer grade products, making it possible to easily apply these models for data analysis.
Within this thesis the light–matter interaction for an hypothetical medium is simulated on the basis of a few-level model. These spectra are used to train and test several machine- and deep-learning models with the goal to reconstruct different pulse characteristics from a simulated absorption spectrum.

The thesis is structured in the following way: In section 2 an overview of fundamental physics needed to describe the light-matter interactions and occurrence of absorption spectra is given. Section 3 covers the models used for analysis and establishes a fundamental catalog of common expressions required to follow all concepts used during the evaluation of the models. The 4^{th} section covers the setup of all numerical experiments and further elaborates on the variations of the toy problem aimed to solve within this thesis. Lastly, we present our findings in section 5 and give a conclusion of this work and outlook for possible improvements in section 6.

2 Theoretical Background - Physics

Throughout this thesis all calculations, simulations and numerical experiments are done using atomic units instead of SI-units.

2.1 Fundamentals of Quantum Dynamics

In non-relativistic quantum mechanics the most fundamental description of a time-dependent quantum system is given by the Schrödinger equation:

$$-i\frac{\partial}{\partial t}\left|\psi(x,t)\right\rangle = \mathcal{H}(t)\left|\psi(x,t)\right\rangle. \tag{2.1}$$

Here, $|\psi(x,y)\rangle$ corresponds to a time-dependent wave function and $\mathcal{H}(t)$ denotes the Hamiltonian describing the system. In a stationary system with time-independent Hamiltion $\mathcal{H}$, equation (2.1) becomes an eigenvalue equation:

$$\mathcal{H}\left|\psi_n\right\rangle = E_n\left|\psi_n\right\rangle, \tag{2.2}$$

where $|\psi_n\rangle$ denotes a set of n orthogonal eigenstates, each associated with an energy E_n. The first energy level E_1 corresponds to the ground state of the system. By representing a state as a superposition of eigenstates, a general solution of the Schrödinger equation can be given by:

$$\left|\psi(x,t)\right\rangle = \sum_{k=1}^{n} c_k(t)\left|\psi_k(x)\right\rangle. \tag{2.3}$$

A time-propagating solution for the general state is then expressed by:

$$\left|\psi(x,t)\right\rangle = \sum_{k=1}^{n} e^{-iE_k t}\left|\psi_k(x)\right\rangle. \tag{2.4}$$

2.2 Ultrashort pulses of light

Maxwell's theory of electromagnetic light propagation revolves around the wave characteristics of light. A space- and time-dependent electric field $\mathbf{E}(\mathbf{x},t)$ contains all information to fully describe such electromagnetic radiation. When regarding invariant, linearly polarized light, the vectorial and spatial properties can be omitted and the electric field reduces to a scalar quantity $E(t)$. Since short light pulses are characterized by a continuum of frequencies, we can give an alternate representation in the frequency domain by using Fourier's theorem:

$$\tilde{E}(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} E(t)e^{i\omega t}dt. \tag{2.5}$$

The function $\tilde{E}(\omega)$ is complex-valued, denoted by the tilde and can be called spectral field. By applying the inverse Fourier transform, the real-valued electric field can be regained:

$$E(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \tilde{E}(\omega)e^{i\omega t}d\omega. \tag{2.6}$$

The spectral field is symmetric around $\omega = 0$ and can generally be written in terms of an amplitude and phase:

$$\tilde{E}(\omega) = \tilde{\mathcal{E}}(\omega)e^{i\tilde{\phi}(\omega)}. \tag{2.7}$$

The spectral phase $\tilde{\phi}(\omega)$ is defined in the frequency domain and can be expanded as a power series into:

$$\tilde{\phi}(\omega) = \phi_0 + \tau(\omega - \omega_0) + b(\omega - \omega_0)^2 + c(\omega - \omega_0)^3 + \dots \tag{2.8}$$

Here ϕ_0 describes a constant offset and τ is called the group delay (gd) which causes a temporal shift of the pulse. The second-order phase term b is known as the group-delay dispersion (gdd) and causes a linear chirp of the pulse. Higher order terms are simply referred to as third order dispersion (TOD) and so on.
Further mathematical descriptions of ultra short pulses of light can be found in text books [2]. In this thesis higher order dispersion effects of $n \geq 4$ are disregarded, since their effects on the pulse shape are minimal.

2.3 Light-matter interactions

To gain insight into the dynamical behavior of atoms or molecules, an external light source can be used to probe the system. If we know the properties of the incoming light source, we can quantify the effects light has on the matter. The absorbance of a sample can be determined by comparing the intensity transmitted through the sample $I(\omega)$ to the incoming intensity $I_0(\omega)$. Typically, the optical density (OD) is used as a measure, which is calculated according to:

$$\mathrm{OD}(\omega) = -\log_{10} \frac{I(\omega)}{I_0(\omega)} \tag{2.9}$$

When regarding linear absorption, Beer-Lambert's law is used to describe the empirically observed drop in intensity:

$$I(\omega, z) = I_0(\omega) \cdot e^{-\alpha(\omega) z}. \tag{2.10}$$

The exponential decay depends on the length of the traversed medium z, as well as on a material-specific absorption coefficient $\alpha(\omega)$. The coefficient is related to the microscopic absorption cross section $\sigma(\omega)$ and the atomic density ρ_N:

$$\alpha(\omega) = \rho_N \cdot \sigma(\omega). \tag{2.11}$$

The cross section can be calculated by [3]:

$$\sigma(\omega) = \frac{\omega}{\epsilon_0 c} \mathrm{Im}\Big(\frac{d(\omega)}{E(\omega)}\Big). \tag{2.12}$$

Here, $E(\omega)$ is the spectral field and $d(\omega)$ the dipole response of the system. Bringing together equations (2.9) and (2.10), we get for the OD:

$$OD(\omega) = \frac{\sigma(\omega)}{\ln 10} \cdot \rho_0 \cdot z. \tag{2.13}$$

Therefore, the optical density is proportional to $\sigma(\omega)$, and the propagation length z.

2.4 Transient absorption spectroscopy

By transmitting light through a target and measuring the electromagnetic field after the transmission, we can gain insights into the quantum dynamical behavior of the system as well. Solving Maxwell's wave equation for the electromagnetic field classically and treating the target quantum

mechanically we obtain a solution for the time-dependent electric field.
In its most principle form, Maxwell's wave equation states:

$$\Big(\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\Big)\boldsymbol{E}(\boldsymbol{x},t) = \frac{4\pi}{c^2}\frac{\partial^2}{\partial t^2}\boldsymbol{P}(\boldsymbol{x},t), \tag{2.14}$$

where $\boldsymbol{P}(\boldsymbol{x},t)$ is the induced polarization of the target. To simplify this equation we can cast the electric fields into plane waves in $z-$direction, so that we may neglect $\frac{\partial^2}{\partial x^2}$ and $\frac{\partial^2}{\partial y^2}$. Furthermore we can assume the target as a very thin layer of matter, such that any propagation effects may be neglected and the target can be treated as a point potential [4]. Maxwell's equation can then be reduce to:

$$\Big(\frac{\partial^2}{\partial z^2} - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\Big)\boldsymbol{E}(z,t) = \frac{4\pi}{c^2}\frac{\partial^2}{\partial t^2}\boldsymbol{P}(t)\delta(z), \tag{2.15}$$

with $\delta(z)$ representing the Dirac delta function. A solution to this equation, assuming a positive propagation in z-direction is then given by:

$$\boldsymbol{E}(z,t) = \boldsymbol{E}_{\text{in}}\Big(t - \frac{z}{c}\Big) + \boldsymbol{E}_{\text{gen}}(z,t), \tag{2.16}$$

where

$$\boldsymbol{E}_{\text{gen}}(z,t) = -\frac{2\pi}{c}\Big[\Theta(z)\frac{\partial}{\partial t}P\Big(t - \frac{z}{c}\Big) + \Theta(-z)\frac{\partial}{\partial t}P\Big(t + \frac{z}{c}\Big)\Big] \tag{2.17}$$

can be obtained after transposing. Here $\boldsymbol{E}_{\text{gen}}$ denotes the interacting electric field, and Θ the Heaviside function.
We can measure the intensity of the light after traversing the medium, which may be expressed as:

$$I(\omega) \propto \Big|\int \boldsymbol{E}(z,t)e^{i\omega t}dt\Big|^2. \tag{2.18}$$

If we use (2.16) and as shown in (2.5), transform the electric field into the fourier domain, we get three terms:

$$I(\omega) \propto |\tilde{\boldsymbol{E}}_{\text{in}}(\omega)|^2 + 2\text{Re}[\tilde{\boldsymbol{E}}^*_{\text{in}}(\omega)\cdot\tilde{\boldsymbol{E}}_{\text{gen}}(\omega)] + |\tilde{\boldsymbol{E}}_{\text{gen}}(\omega)|^2. \tag{2.19}$$

We further assume the incoming light field strength to be much greater than the interacting parts of the light $\tilde{\boldsymbol{E}}_{\text{in}}(\omega) \gg \tilde{\boldsymbol{E}}_{\text{gen}}(\omega)$, such that the last term in (2.19) can be dropped:

$$I(\omega) \propto |\tilde{\boldsymbol{E}}_{\text{in}}(\omega)|^2 + 2\text{Re}[\tilde{\boldsymbol{E}}^*_{\text{in}}(\omega)\cdot\tilde{\boldsymbol{E}}_{\text{gen}}(\omega)]. \tag{2.20}$$

With this equation we can describe the observed absorption spectrum in terms of merging electric fields.

2.5 Few-Level Model

For the study of quantum systems deemed too complicated for analytical solutions, numerical simulations can help to provide insights into the inner dynamics of such a system. To implement simulations, one must first conceptualize models that describe the major features of a quantum system. The following sections are intended to act as an introduction to the basic concepts of a Few-Level Model and its numerical solution.
For studies regarding the dynamics of light-matter interactions such as ultrashort laser pulses in simple quantum systems, it is advantageous to separate the total Hamiltonian $\mathcal{H}$ into two parts:

$$\mathcal{H}(t) = \mathcal{H}_0 + \mathcal{H}_{int}(t). \tag{2.21}$$

The first operator $\mathcal{H}_0$ describes an unperturbed, time-independent system, whereas $\mathcal{H}_{int}$ denotes the light-matter interacting part of the system. Since all systems we consider can be described by a finite number of states, we can expand the wave function into a finite set of energy eigenstates of the unperturbed system:

$$|\psi(t)\rangle = \sum_{k=1}^{n} c_k(t) |k\rangle , \tag{2.22}$$

where $c_k(t)$ are complex-valued, time-dependent state coefficients. The time evolution of such a system can be found by expressing the state in its eigenbasis. By projecting $\langle\psi(t)|$ onto $|\psi\rangle$, we can depict the Hamiltionian by a $n \times n$ matrix, with energies $E_k = \langle k| \mathcal{H}_0 |k\rangle$ on the diagonal and couplings between states $D_{kl} = \langle k| \mathcal{H}_{int}(t) |l\rangle$ on the off-diagonal:

$$\mathcal{H} = \begin{pmatrix} E_1 & D_{12} & \dots & D_{1n} \\ D_{21} & E_2 & & D_{2n} \\ \vdots & & \ddots & \vdots \\ D_{n1} & D_{n2} & \dots & E_n \end{pmatrix}, \qquad |\psi\rangle = \begin{pmatrix} c_1(t) \\ c_2(t) \\ \vdots \\ c_3(t) \end{pmatrix}. \tag{2.23}$$

Using dipole approximation, the interacting part can be described by $\mathcal{H}_{int} = \hat{d} \cdot E(t)$, where $\hat{d}$ and $E(t)$ represent the dipole moment and electric field. Therefore the matrix elements D_{kl} can be evaluated as:

$$D_{kl} = \langle k| \mathcal{H}_{int}(t) |l\rangle = \langle k| \hat{d} |l\rangle \cdot E(t). \tag{2.24}$$

A formal solution for the Schrödinger equation implementing (2.23) is:

$$|\psi(t)\rangle = e^{-i\mathcal{H}t} |\psi_0\rangle , \tag{2.25}$$

with $|\psi_0\rangle = |\psi(t=0)\rangle$ representing the initial wavefunction.

2.6 Numerical Solution

For a diagonal operator, the matrix exponentials are reduced to a diagonal matrix of exponentials of the matrix elements. When only accounting for a field-free propagation, the solution (2.25) can therefore be reduced to a simple phase evolution of each eigenstate. For the interacting parts we transform into the eigensystem of $\mathcal{H}$ such that it can be represented in a diagonal form.
The interacting part of the Hamiltonian is explicitly time-dependent, so that the evolution of the coefficients $c_i(t)$ needs to be performed in discrete time steps Δt. With this approximation we assume that for a sufficiently small time step the propagation can be regarded as constant.
With $t_k = k\Delta t$ denoting the total propagation time, each iteration is split into the following sequence of steps [5]:

1. Field-free evolution:
$$c_i(t + \Delta t) = e^{-iE_i \Delta t} c_i(t)$$

2. Transformation into the eigensystem of the Hamiltonian:
$$\tilde{\boldsymbol{V}} = \mathrm{diag}(\tilde{V}_1 ... \tilde{V}_n) = T^\dagger \hat{\boldsymbol{V}} T, \quad |\tilde{\psi}\rangle = T^\dagger |\psi\rangle .$$
Here the tilde denotes the switch of basis, which is accomplished by taking the eigenvectors of $\tilde{\boldsymbol{V}}$ as transformation matrices T.

3. Propagation in the interacting domain, denoted by a tilde:

$$\tilde{c}_i(t + \Delta t) = e^{-i\tilde{V}_i \Delta t} \tilde{c}_i(t)$$

4. Transforming back into the energy basis:

$$|\psi\rangle = T\,|\tilde{\psi}\rangle$$

For computational reason all numeric calculations are done using atomic units.

3 Theoretical Background - Machine Learning

3.1 Supervised Learning and Regression

Supervised Learning (SL) is a major branch of machine learning. By mapping input to output based on training examples SL can learn regularities in a given set of data. Each training example consists of an input object (features) paired with a desired output value (targets). A supervised learning algorithm learns on the training data to approximate an inferred function, which can then be used on new unseen examples. In short, SL can thereby be understood as the estimation of the inferred function between features and targets by learning the characteristics of training examples.
In statistics, regression analysis is a set of mathematical tools to estimate the dependencies between response variables and independent observed variables. The main difference to the well known classification models is the ability to predict continuous variables, but both are widely used in machine learning.

3.2 Fundamental Models

3.2.1 Linear Regression

The most basic regression model expects the target values to be a linear combination of the observed features. A prediction $\hat{y}$ would then be calculated by

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \cdots + w_n x_n \tag{3.1}$$

where $w = (w_0, \ldots, w_n)$ designates the fitted weight vector. For a given set of data pairs (X, y) where X denotes the independent observations and y the dependent variables, the model can be computed by minimizing the residual sum of squares between the observation and responses:

$$\min_w ||Xw - y||_2^2 \tag{3.2}$$

The actual implementation is done in python with existing libraries, in this case with *sklearn* [6].

3.2.2 Decision Tree

Another fundamental method used in supervised learning for regression and classification are the so-called Decision Trees. By learning simple decision rules inferred by features of the training data, a tree can learn simple patterns very fast. Mathematically a decision tree can be seen as a piece-wise constant approximation function.

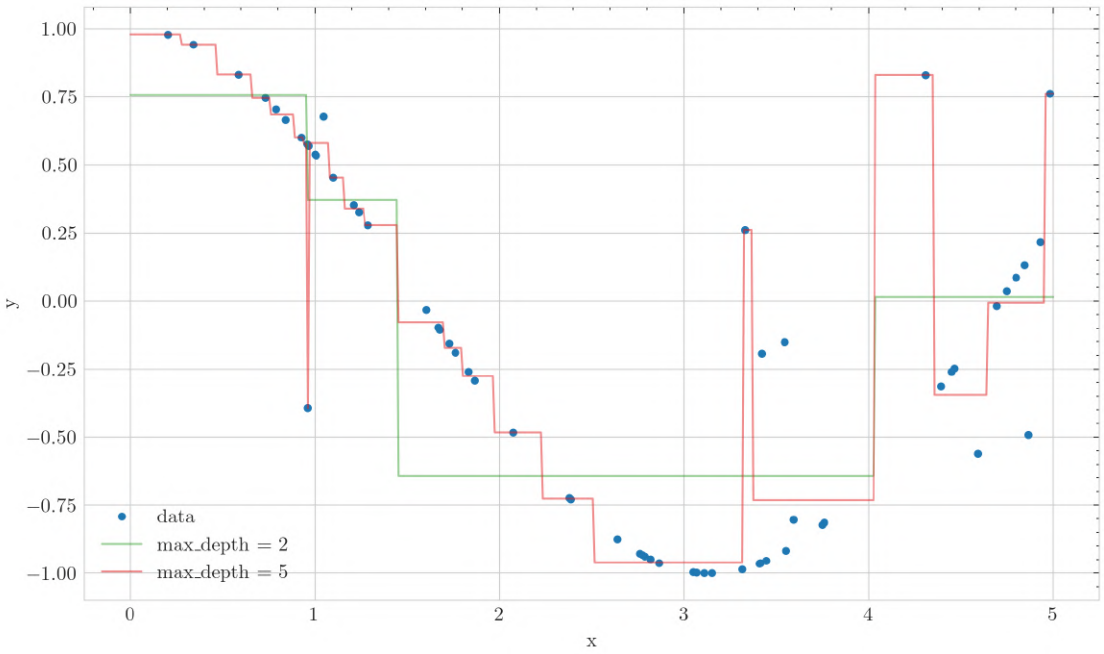


Figure 3.1: Decision Tree Regression: The expectation value y is shown as a function of the input values x for both the data and two different decision trees. In green, a decision tree regressor with a maximum tree depth of two can be seen. It overfits less, but is not as accurate. In red, the decision boundaries of a tree with a maximum depth of five nodes are visualized. It is much more accurate, but where data points are more spread out, overfits significantly.

The structure of a decision tree has two main components, decision nodes and leaf nodes as depicted in Figure 3.2.

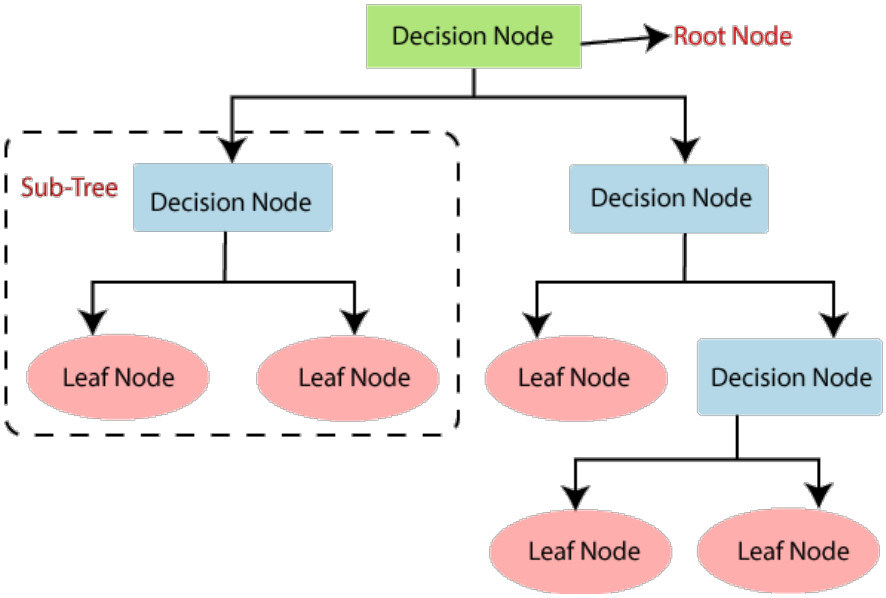


Figure 3.2: The basic structure of a decision tree as described in [7]. All nodes at the end of the tree are called leaf nodes.

Some of the advantages and disadvantages of tree regressors are listed in Table 3.1:

Pros	Cons
Computationally cheap, because it only scales logarithmic with dataset size. Since they are simple to understand and interpret, they can be visualized as well.	Prone to overfitting by creating overly complex trees, which do not generalize well. This however can be countered by limiting the maximum depth of the tree or setting a minimum number of samples per leaf (the final node of a branch). Trees are often unstable since small variations in the data can result in entirely different decision rules being created. This can be mitigated by using multiple trees in an ensemble, also called a decision forest. If unbalanced data is used, biased trees can emerge resulting in faulty models.

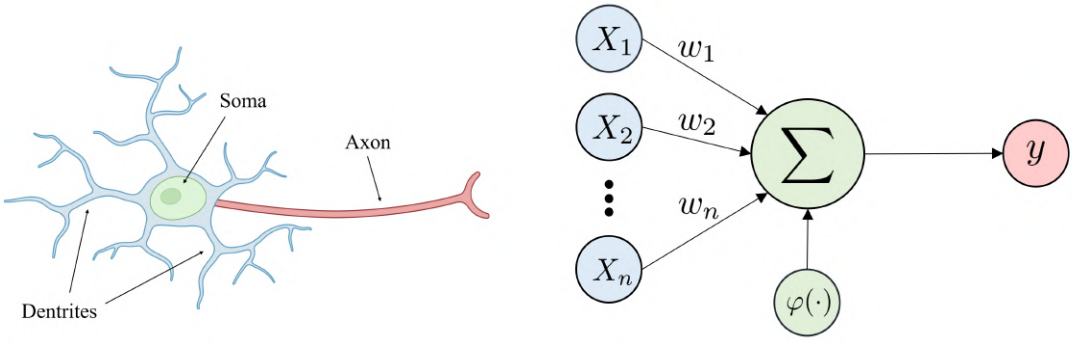
Table 3.1: Advantages and disadvantages of decision trees

3.3 Deep Learning Models

Deep Learning is a subcategory of machine learning, utilizing artificial neural networks. In contrary to conventional machine learning algorithms neural networks have the potential to solve the most complex of problems, by providing many different easily scale-able model architectures to match the difficulty of a problem.
The following sections give a short introduction into basic machine learning expressions as well as two of the most common types of neural networks and how they work.

3.3.1 A basic neural network

Perceptron
The most basic building block of any neural network is the perceptron, which is a well known function to transmit signals y depending on the input **X** as described below. Inspired by the biological neuron found in the brain, the perceptron is modeled along the same working principles, to receive, process and transmit signals:



(a) A biological neuron **(b)** A mathematical perceptron

Figure 3.3: **(a)** A depiction of biological neurons found in brains. The three most important features are the cell body (soma) to process information, branched extensions that propagate electric stimulation (dentrites) and an axon to transmit information away from the neuron. **(b)** The basic structure of a perceptron. It is modeled very similar to a neuron.

Mathematically the output y of a perceptron is calculated by:

$$y = \varphi(\mathbf{X} \cdot \boldsymbol{\omega} + b), \tag{3.3}$$

where $\mathbf{X}$ represents an input vector of features X_i, $\boldsymbol{\omega}$ a weight vector, b the so-called intercept and φ the activation function.
The intercept is used to shift the activation to account for bias in the training data. To be able to map non-linearities or to restrict the span of the output, a non linear activation function is needed.

Activation functions
A few popular activation functions are depicted in Figure 3.4, from left to right:

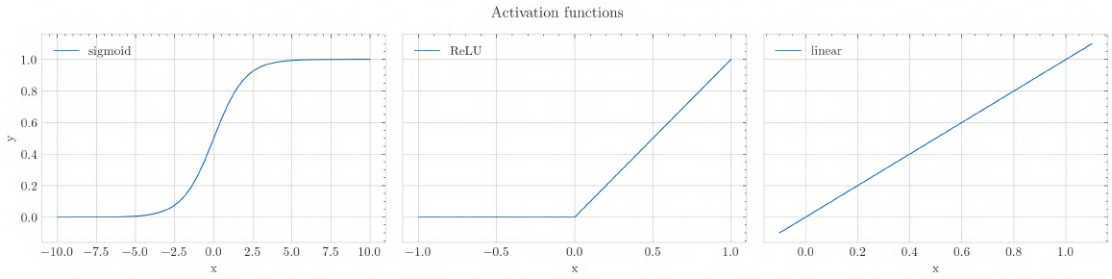


Figure 3.4: Activation functions commonly found in deep learning models

- The sigmoid function has been the most prominent activation function in early neural networks. It is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{3.4}$$

  however more computationally advantageous functions have replaced it more recently.

- The Rectified Linear Unit (ReLU) stands out for its simplicity. In its most basic form it is defined as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3.5}$$

  and hence, can be computed very cheaply. It has replaced the sigmoid as the most frequently used activation function.

- The linear function acts as a pass-through function and is commonly used to deregulate the output and allow negative outputs to be calculated.

- Another activation function, frequently used in classification tasks, models the output into a probability distribution and is called soft-max function:

$$f(x) = \frac{e^x}{\sum_i e^{x_i}}. \tag{3.6}$$

  For multiple outputs their sum will be equal to 1 and each value can be interpreted as a probability. Note, that this function has not been used in the subsequent experiments, since only regression problems were considered.

Layers and neural network

By organizing multiple perceptrons in parallel, a **layer** can be formed and by cascading multiple layers back to back a **neural network** is created. The first and last layer are described as input and output layers, while all layers in between are called hidden layers:

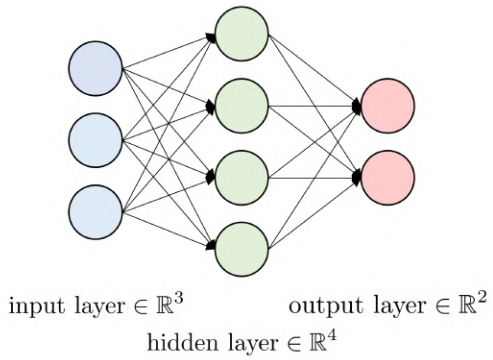


Figure 3.5: A small neural network. Every dot is representing a single perceptron, while each vertical column of dots represents a layer. This network has 3 inputs, 1 hidden layer with 4 neurons and 2 outputs.

For a neural network as depicted in Figure 3.5 a prediction $\hat{y}$ can be calculated by feeding the output of the first layer into the second layer as the input and so on, resulting in:

$$\hat{\boldsymbol{y}} = \varphi_2(\boldsymbol{X_2} \cdot \boldsymbol{w_2} + \boldsymbol{b_2}) = \varphi_2(\varphi_1(\boldsymbol{X} \cdot \boldsymbol{w}_1 + \boldsymbol{b_1}) \cdot \boldsymbol{w_2} + \boldsymbol{b_2}). \tag{3.7}$$

Here $\boldsymbol{w}_i$ denote weight matrices and $\boldsymbol{b}_i$ represent intercept vectors.

3.3.2 Training expressions

With help of equation (3.7) a formal way was introduced to connect input values $\mathbf{X}$ to the desired output y. The main goal of the neural network is to determine the output as precise as possible by means of varying the weights ω. For this a so-called loss function will be introduced which formalizes the learning process of the network, so that it can be optimized on initial data set – which in turn is known as the training process.

Loss function
A loss function, commonly referred to as a cost function, calculates the deviation of a prediction to the true expected value. Depending on the task, different loss functions are better suited [8]:

- Mean Squared Error (MSE)/ L2 Loss:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

  with $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ denoting the true value an predicted value of a model. This loss is used for regression as well as classification.

- Absolute Error (AE)/ L1 Loss:

$$AE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|,$$

  Similar to mean squared error, but less used in literature.

- Log Loss for Logistic Regression:

$$LogLoss = \sum_{i=1}^{n} -y \log(\hat{y}_i) - (1-y) \log(1-\hat{y}_i).$$

  A loss function modeled for logistic regression. Labels y must be either 0 or 1 and predictions are between 0 and 1.

Backpropagation
A neural network learns by using an algorithm called backpropagation. Explaining the entire algorithm is beyond the scope of this thesis, but a short summary will be given nevertheless:
First a model gets initialized with random weight vectors. This has to be done since the best set of weights is yet unknown. For an input vector, the prediction is calculated and compared to the desired output, which must be and in fact is known for the training set[1]. By calculating the loss, a quantitative measure of quality of the prediction can be given. The weights are then adjusted by using gradient descent [9] to minimize the loss. This is repeated multiple times until the loss converges and no longer improves [10].
For the gradients, numerous derivatives have to be calculated. This task gets very hard for larger networks, thus automatic differentiation [11] is used in practical applications.

Learning rate and optimizer
The step size, when using gradient descent to update the weights, dictates how fast a model converges to an optimal solution. The step size is commonly referred to as **learning rate**. When the steps are too large, the model might not converge at all and if they are too small, the gradient might get stuck in a local minimum. Therefore, it is advantageous to use a variable learning rate e.g. exponential decay. Here, the learning rate decreases exponentially, depending on how many update steps have been completed so far.
The **optimizer** specifies the algorithm used to calculate the gradient direction. It is not always best to descend into the direction of steepest gradient. Sometimes it can be useful to introduce randomness (stochastic gradient descent), or include the second derivative (Newton/Raphson). Currently the most successful and universally applicable algorithm, is the adaptive moment estimation (ADAM) optimizer [12]. Though every few years, a new best algorithm appears to be popularized.

Batch and Epoch
Computationally it is not advantageous to compute the gradient and update the parameters for each training example successively. We can improve the training process by taking multiple samples together to form a bigger training **batch** and updating the model for such a batch. For a batch the model can be updated by using tensor products for predicting and averaging or summing the gradient for backpropagation [10].
For decision trees, linear regression and similar simpler models, there are no benefits on fitting the same identical data multiple times. The resulting model would not change, or change only slightly in case of induced randomness. However, for deep learning models, which rely on parameter updates via gradient descent, it does make a difference to learn on the same data repeatedly. One pass through the entire training data, is called an **epoch**. Depending on the problem and model it can take tens to thousands, or for very complex problem millions of epochs of training, for a model to converge.

[1] This is also why networks do not work without proper training (data).

3.3.3 MLP - Multi-Layer Perceptron

A fully connected (dense), neural network with at least one hidden layer is commonly referred to as *Multi-Layer Perceptron* (MLP). It is categorized as a feedforward neural network with no cyclic connections [13] and is one of, if not the most, versatile neural network architecture to this day. In this thesis, this model is used conjointly with some variations of it to conduct numerical experiments. Two of these alternations, mainly to reduce overfitting, are outlined in the subsequent paragraphs.

Regularization
There are many ways to introduce regularization into a deep learning model. A common approach for regularizing a model, is by incorporating an additional term into the loss function:

- L1-Regularization

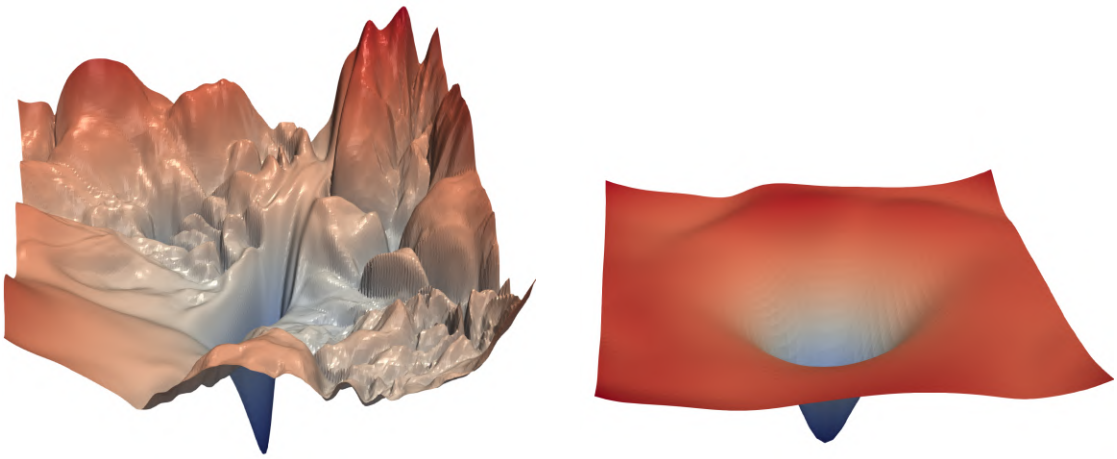
$$Loss = Loss\ function + \lambda \sum_{j=1}^{n} |\theta_j|$$

- L2-Regularization

$$Loss = Loss\ function + \lambda \sum_{j=1}^{n} \theta_j^2$$

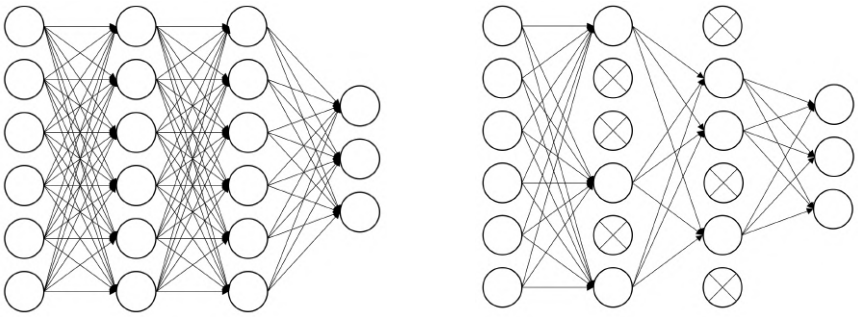
Here θ_j denote the weights of the model, which means regularization acts as a penalty on the weight values. Note that not all weights of all layers have to be used when building a model. For different layers, different regularization terms, and parameters λ may be chosen. In general it can be difficult to choose the correct parameter λ. If chosen to large, the regularization term outweigh the loss function and the actual data has minimal effect on learning and when chosen to small the opposite may happen.
Regularization has the effect of smoothing the loss-landscape, i.e. the space in which the gradient search is being conducted. Since weights are penalized, smaller weights are preferable. With smaller weights, small input variations are expected to have less impact on outcome, resulting in a more stable model.
Loss-landscapes cannot be visualized generally, however an illustration can be given, as shown in the Figure 3.6:

Dropout Layers
Another approach to induce regularization are so-called dropout layers. During training, nodes in layers are disabled at random. This results in a model which can generalize better, since multiple combinations of weights are driven to produce the same prediction.



(a) without regularization **(b)** with regularization

Figure 3.6: The loss landscape without **(a)** and with **(b)** regularization, reprinted from NeurIPS 2018 [14], depicting the impact of skip connections as regularization on the loss landscape of ResNet-56. Note: skip connections are a different kind of regularization, mainly aimed at solving the vanishing-gradient problem in deep neural networks [15].



(a) without dropout layers **(b)** with dropout layers

Figure 3.7: A visualization of a dropout layers in a dense feedforward neural network with one input, two hidden and one output layer. No output nodes should be disabled.

3.3.4 CNN - Convolutional Neural Network

A convolution is a mathematical operation defined by:

$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau. \tag{3.8}$$

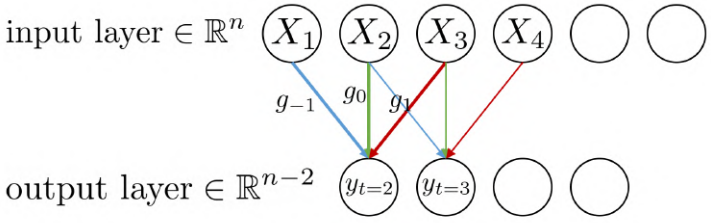
Convolutional layers implement a discrete and finite variation of correlation closely related to convolution. So let f and g be defined on $\mathbb{Z}$, the finite discrete convolution is given then by:

$$(f * g)[t] = \sum_{\tau=-T}^{T} f[t-\tau]g[\tau]. \tag{3.9}$$

This convolution can be interpreted as a discrete filter operation g on a given input signal f. Here T are called *taps*, and $(2T+1)$ is the receptive field or window of the filter.
The correlation actually implemented is then given by:

$$(f \star g)[t] = \sum_{\tau=-T}^{T} f[\tau + t]g[\tau]. \tag{3.10}$$

A visualization of a single 1D-convolutional layer, with a filter moving over an input signal can be regarded as shown in Figure 3.8:



$$y_t = (X \star g)[t] = \sum_{\tau=-1}^{1} X[\tau + t]g[\tau] = X_{t-1}g_{-1} + X_t g_0 + X_{t+1}g_1$$

Figure 3.8: A visualization of a 1D-convolutional layer with one filter moving across. The same weights are are used for all outputs, only the input changes. Note that the output dimension is reduced by two times the number of taps, in this example the first and last output would be disregarded.

Some important expressions, when dealing with convolutional layers in neural networks are:

- In the context of neural networks **kernel** is an equivalent designation for filter. This is not the case in general. Kernel size refers to the receptive field of a filter, which means how many of the input signals are taken into account for the convolution. In Figure 3.8 the kernel size is 3.
- Stacking multiple filters results in a collective of **channels**. Similarly an input can have multiple channels e.g. RGB for images.
- **Stride** denotes the distance or number of skipped inputs, when moving a filter across the input vector, e.g. in Figure 3.8 a stride of 1 is used for illustrating the convolution.
- How to handle the margins of a convolution, is referred to as **padding**. A valid padding disregards the incomplete outputs, first and last node $y_{t=1}$ and $y_{t=N}$ in Figure 3.8, while zero padding fills the missing inputs with zeros. This however has dimming effects at the borders and is thus not as widely used.

3.4 Training practices

Data preparation
Besides the usual data preparation steps like dropping incomplete sets, one major improvement for performance can be scaling the input data and labels. Especially for regression problems where some parameters have different scales, it is a necessity to scale all regression labels to be

in the same magnitude. If this was not the case, the gradient would vanish for much smaller target values.

Training and Test Set
To be able to train and test a model on a singular dataset, it is common practice to split the data into a training and testing set. The model can then be trained on the training set and validated afterwards on the unseen test data.

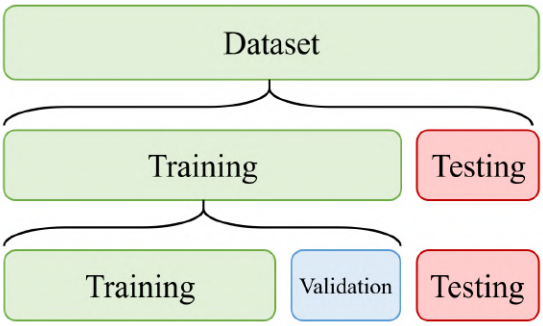


Figure 3.9: Graphical representation of splitting data into a training, validation and testing set. If the dataset consists of very few samples, k-fold cross validation may be used instead [16].

This practice is found in all of machine learning and data science in general. For deep learning models, where training generally takes much longer, a portion of the training data set gets separated into a validation set. Like test data, this set is not used for training, but rather for monitoring the training progress. During training the model gets validated on unseen data, and in case of overfitting, the training could be stopped early.

4 Numerical Experiment Setup

4.1 Simulations

The foundation on which this thesis operates, is a numerical few-level simulation provided by Prof. Dr. Thomas Pfeifer.
The simulation generates a synthetic absorption spectrum on the basis of the numerical solution for a few-level model. For a given hamiltonian a numerical absorption spectrum is generated, using the desired pule frequency and shape.
Based on this infrastructure, a mechanism to save all information which went into and came out of the simulation was added. To have more potential training data, the optical density for each spectrum was calculated and saved as well.
In total, 100.000 simulations have been run and saved, where the following parameters were held constant:

Parameter	Value	SI-units
Δt [atomic time unit]	1	24.2 as
steps [Δt]	4096	99.1 fs
steps Cont [Δt]	28672	700.5 fs
frequency [$\frac{1}{\Delta t}$]	$\frac{1}{190}$	0.14 eV
spectral width [$\frac{1}{\Delta t}$]	$\frac{1}{30}$	0.9 eV
pulse delay [Δt]	2048	49.5 fs
OD scaler [$n.u.$]	5	

Table 4.1: Parameters held constant during simulation. Here, the time step size Δt , the number of steps needed for the pulse propagation "steps" and the number of steps to evolve after the pulse is over"steps Cont" define the framework of the simulation. "pulse delay" ensures the pulse is centered within the steps and "OD scaler" scales the optical density.

Parameter	Uniform range
fieldstrength [a.u.]	$[0, 1]$
2nd order phase [rad/fs]	$[\pm 0.01]$
3rd order phase [rad/fs]	$[\pm 0.0002]$

Table 4.2: Parameters and intervals of which uniformly sampled combination were used to generate the absorption spectra.

Variations of the fieldstrength result in a non-linear increase of intensity in the absorption spectra, while 2nd- and 3rd-order phase variations produce differently shaped pulses as depicted in Figure 4.2 and Figure 4.1.
The absorption spectra is mathematically given by:

$$I(\omega) \propto |\frac{1}{\mathrm{a}}\tilde{\boldsymbol{E}}_{\mathrm{gen}}(\omega) + \tilde{\mathcal{E}}(\omega)e^{(-\frac{\omega-f}{\mathrm{sw}})+b\cdot(\omega-f)^2+c\cdot(\omega-f)^3}|^2. \tag{4.1}$$

Here a,b,c refer to the three varied pulse parameters, fieldstrength, quadratic chirp or 2nd-order spectral phase and 3rd-order spectral phase respectively. The frequency f and spectralwidth sw are held constant.

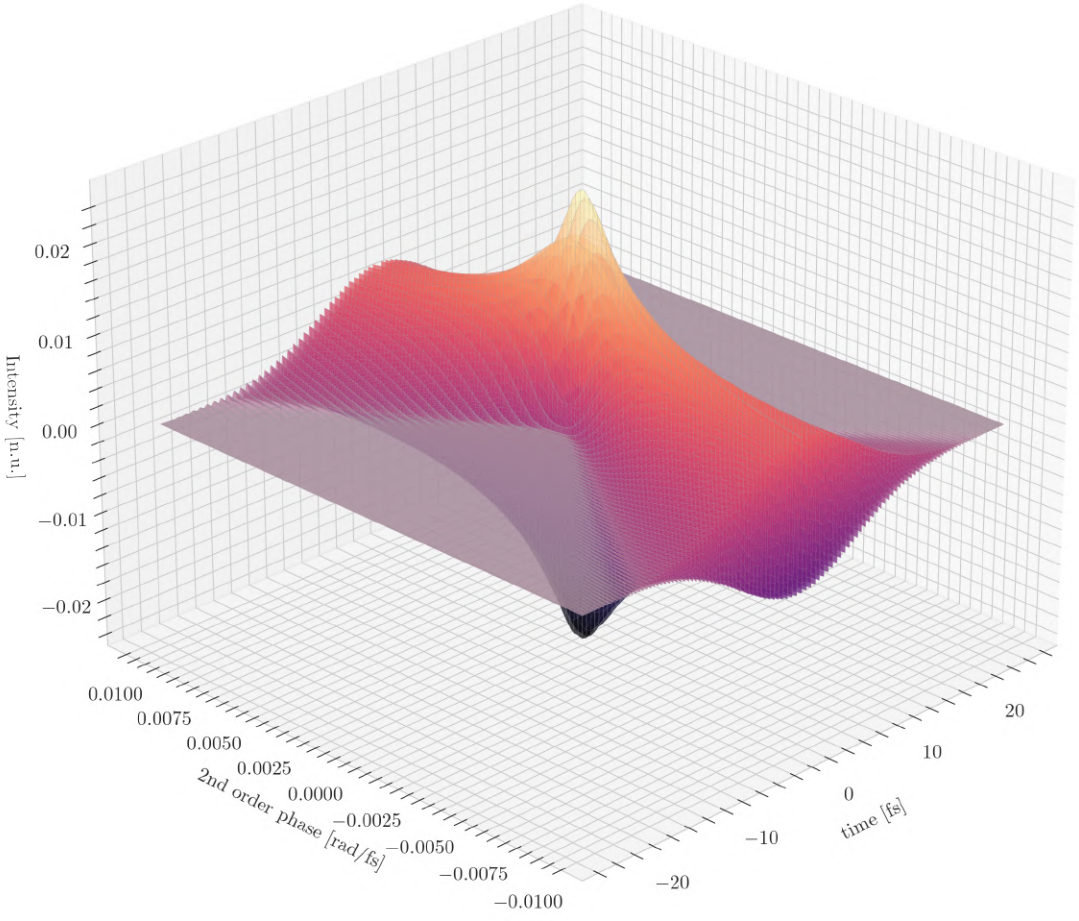
Even simple atoms and molecules are described by many energy levels. This leads to a large Hamiltonian describing the system, which usually results in long simulation times. It is not the goal of this thesis to recreate measured spectra, but rather to explore the viability of machine learning to extract pulse parameters from the input of an absorption spectrum. Therefore an

arbitrary, not too easy, but also not too complex system was chosen, where all excited states couple to the ground state but not to each other:

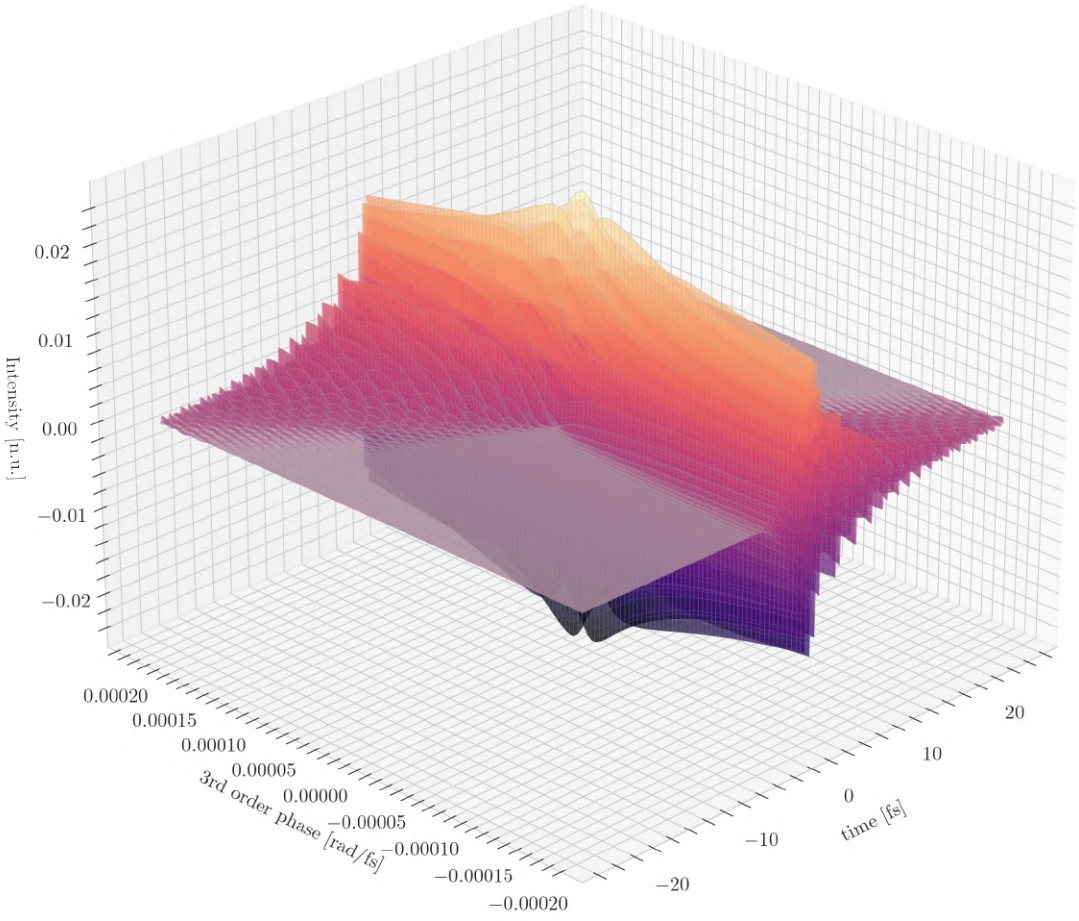
$$\mathcal{H}_{free} = \begin{pmatrix} 0+0i & 0 & \cdots & 0 \\ 0 & 0.2-0.001i & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 0.4-0.001i \end{pmatrix}, \quad \mathcal{H}_{int} = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & & 0 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}$$

The resulting Hamiltonian can be represented as a (20×20) complex-valued matrix.

The change of pulse shape induced by varying the 2nd- and 3rd- order phase terms are visualized in Figure 4.1.



(a) Variation of 2nd-order phase



(b) Variation of 3rd-order phase

Figure 4.1: The pulse form during variation of **(a)** 2nd-order phase and **(b)** 3rd-order phase parameters in the time domain. The other parameter is set to 0 respectively. A widening in pulse shape can be observed when increasing the absolute 2nd-order phase parameter. And if the 3rd order is increased, a non-linear pulse shape transformation can be seen.

The impact of fieldstrength on the absorption spectra can be visualized as well, as seen in Figure 4.2:

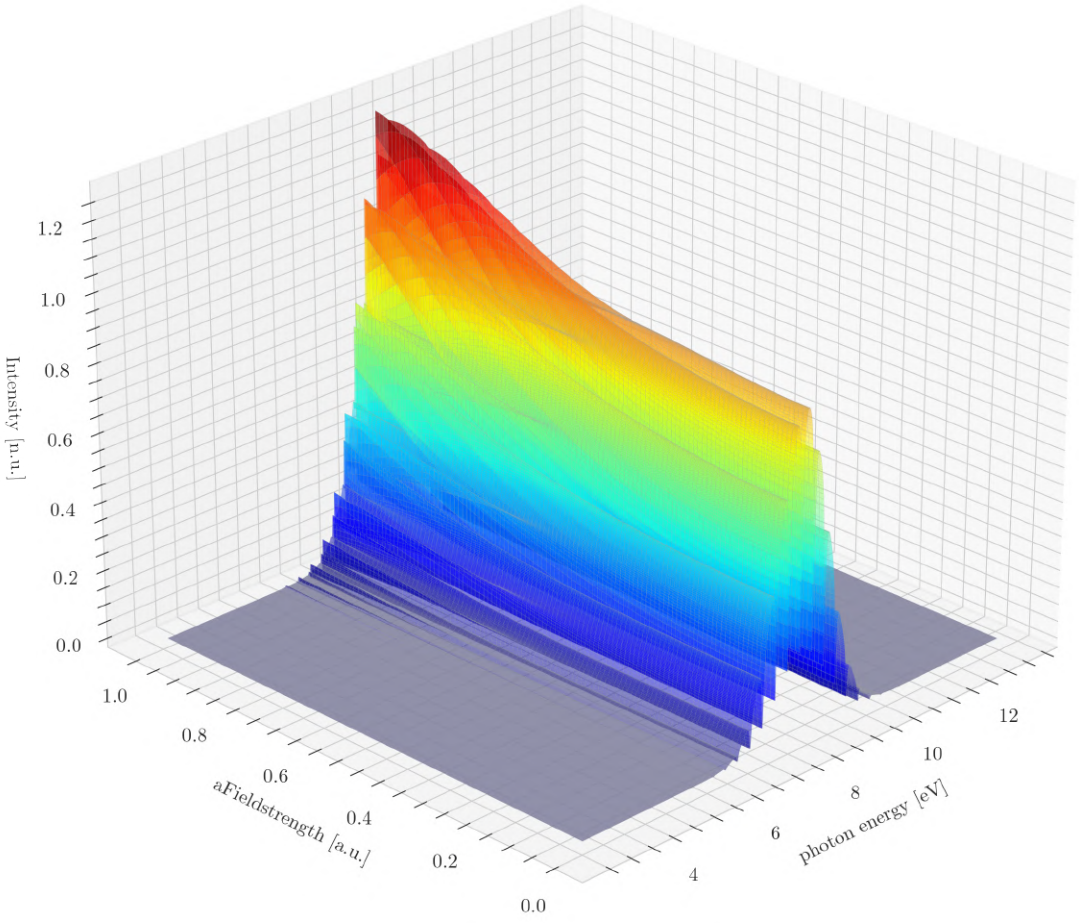


Figure 4.2: The impact of variation of the fieldstrength in the boundaries of Table 4.2 on the absorption spectra in the spectral domain. The spectral phase terms are set to 0. As expected, the intensity rises with an increase in fieldstrength.

4.2 Absorption spectra with noise

To increase the difficulty of the task and to more closely resemble actual experimental data, we further add noise to the simulated spectra.
Three appropriate noise levels were chosen,1%,3% and 5% of the spectra's maximum peak intensity. In this context 1% noise means; the standard deviation of Gaussian noise is 1% of the peak intensity of all spectra. The maximum intensity of all spectra was chosen, to ensure equal noise levels in all spectra.
A randomly sampled spectrum and the three noise levels are shown in Figure 4.3.

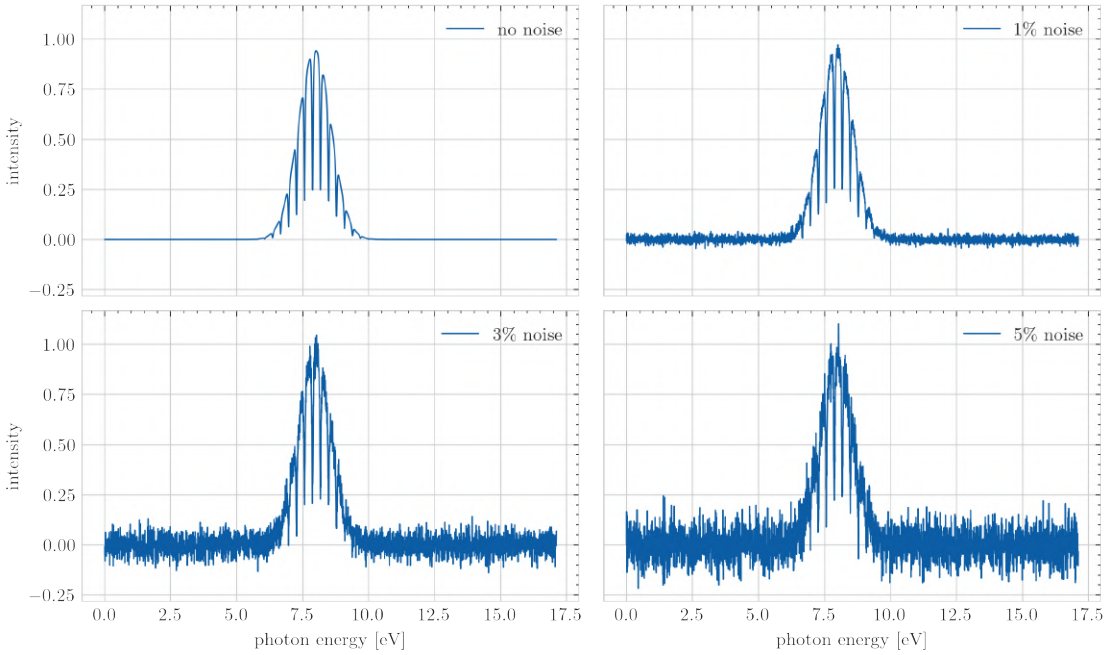


Figure 4.3: An example spectrum with different levels of noise. The spectrum was generated using the Hamiltonian and constant parameters as in section (4.1) and the random parameters were: fieldstrength ≈ 0.28 $[a.u.]$, 2nd order phase ≈ -0.009 $[rad/fs]$ and 3rd order phase $\approx 1.6 \times 10^{-5}$ $[rad/fs]$

5 Results

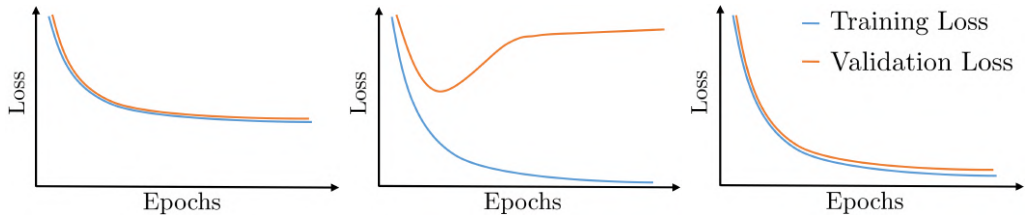
This section covers the evaluation of viability of machine learning algorithms in absorption spectroscopy. The toy problem we aim to solve, is the reconstruction of fieldstrength as well as 2nd- and 3rd-order spectral phase of the pulse from simulated absorption spectra. The in chapter three explored machine learning models were implemented, trained and evaluated using the following metrics.

5.1 Evaluation metrics

For all models which do not have to be trained iteratively, only the fitted model can be evaluated. For deep learning models however, the training progress can be monitored and visualized, giving interesting insights into the model as well as the underlying data.

5.1.1 Evaluating training

To visualize the training progress, it is customary to plot the training and validation loss (section 3.4) over the epochs completed so far. The absolute loss is largely dependent on the dataset and loss function. The relative loss-curve shape and especially the deviations between training and validation loss, give insight about over- and underfitting during training.



(a) Underfit **(b)** Overfit **(c)** Good fit

Figure 5.1: **(a)** The model is underfitting, if both training and validation loss are not converging to zero. **(b)** Overfitting occurs if the training error converges to zero, but the validation error starts to climb or stops converging. This means the model has learned to memorize the entire training set and the associated labels. **(c)** The perfect fit is on the border between over- and underfitting. It is hard to differentiate between a good fit and underfitting of a model.

In some cases, it is unavoidable to encounter overfitting. Typically it is unknown after how many epochs of training the validation error begins to rise again. Therefore, a method to save the model at the moment of best performance is needed.
One option is to stop the training, as soon as the validation loss is no longer improving. This, however, may result in erroneous early stopping and is hence not recommended.
The better alternative is setting checkpoints during training by saving the models weights for every epoch. However, for models with hundreds of thousands of parameters, the entire model cannot be saved thousands of times without running into storage problems. This can be circumvented by saving the model when it performs best, and overwriting the save every time the model improves.

5.1.2 Model evaluation

R^2-Score

The coefficient of determination, usually denoted as R^2, describes the proportion of variance of the predicted variables, that has been explained by the target variables [17].
The score provides a measure of goodness-of-fit and hence an indication how good the model is expected to perform on unseen data.
For a given target data set $\{y - i\}_{i=1...n}$, where each value is associated with a prediction $\{\hat{y}_i\}_{i=1...n}$ a general definition for the coefficient of determination is given by:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{5.1}$$

where the residual sum of squares $SS_{res} = \sum_i (y_i - \hat{y}_i)^2$ and the total sum of squares $SS_{tot} = \sum_i (y_i - \bar{y})^2$ with $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ is the mean of the observed data.
If a regression model perfectly fits the data, the expected R^2 is equal to 1, while the lower bound is usually 0. However, negative values can occur when the model predicts worse than the worst possible least-squares predictor (the null hypothesis, equivalent to a horizontal hyperplane at height of the mean of all data points). This can happen if the wrong model was chosen or illogical constraints are applied.
For vectorial regression, each output class is assigned an individual R^2-Score. The total score is an average of all classes.

Error distributions

The R^2 by itself only gives information about the quality-of-fit of all predictions. To gain further insight, it can be helpful to visualize the error distributions and graphing the predicted values against the true values.

5.2 Results

All diagrams and scores were calculated using scaled parameters. For scaling, the uniform target distributions (section 4.1) were scaled to have a mean of zero and a standard deviation of one. Scaling the targets is advantageous for the visual comparison of the results, while the R^2-score is not influenced by scaling.
The training was conducted on 80% of 10.000 generated spectra, of which further 20% were used to calculate the validation error for all iterative learning models. For deep learning models this results in a total of 6400 samples for training, 1600 samples for validation and 2000 samples for testing. For the basic models the same split was implemented, but including the validation set during fitting, this results in a total of 8000 samples for training and 2000 for testing.

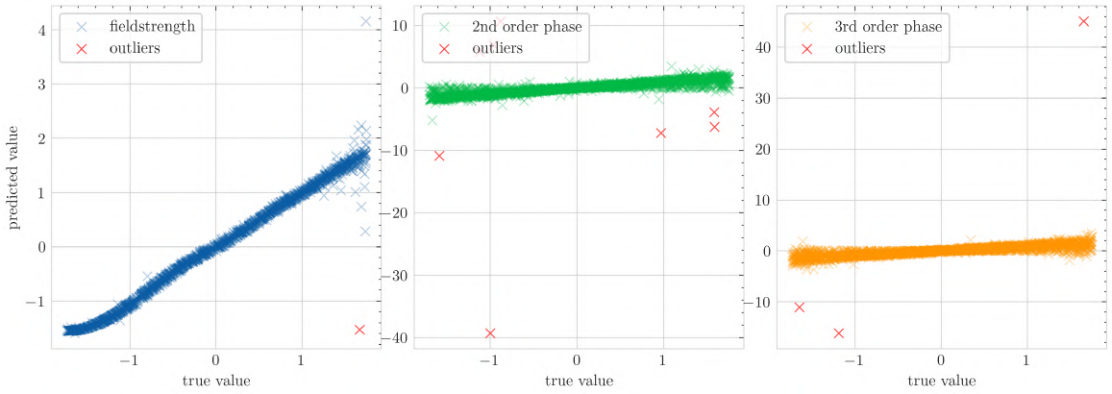
5.2.1 Linear Regression

We start of by evaluating the simplest of all models, the linear regression model. Since all of the data only has to be fitted once, there is no need to inspect the training progress. Only the final model is evaluated. The results for the R^2-score can be found in Table 5.1:

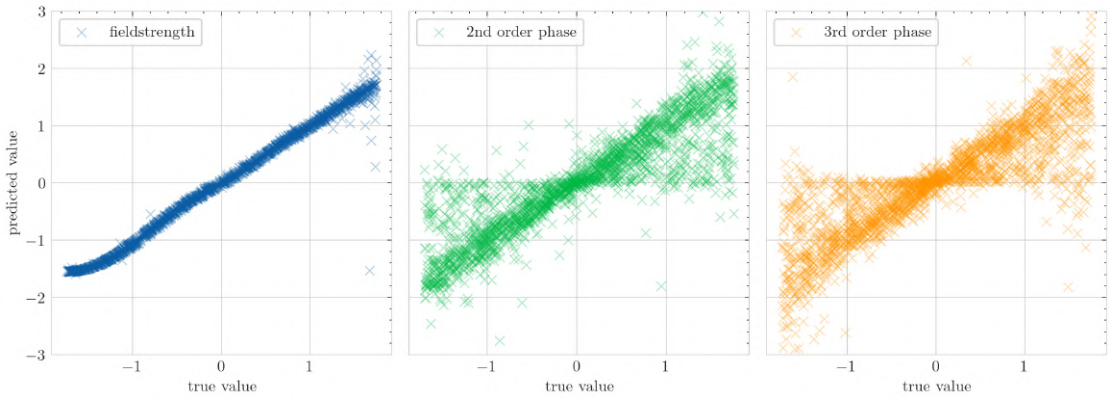
model	fieldstrength	2nd order phase	3rd order phase	mean score
linear regression	0.986690	-0.209403	-0.403350	0.1246

Table 5.1: R^2-scores for linear regression fitted on spectra without noise. The fieldstrength can be reconstructed, while the phase parameters can not.

The model performs very good in determining the fieldstrength of a spectrum, however for the 2nd- and 3rd-order phase it is under-performing significantly. We try to approximate a quadratic and cubic phase relation with a linear function, which apparently does not work.
We can visualize the error distributions, by charting the predicted values against the true values:



(a) true distributions



(b) outliers removed

Figure 5.2: Predicted values plotted against true values. The best fit would be a diagonal line. Although the R^2-score seems to suggest a much worse model, we can clearly see a linear trend in the 2nd- and 3rd-order phase terms.

The seemingly negative correlation can be explained by the very few, but very large outliers, when calculating the R^2-score. This shows a weakness of such metrics since large but very few variations can lead to large differences in supposed model performances.

5.2.2 Decision Tree

Next we inspect a decision tree model. We, again, use the spectra without noise and fit a decision tree with no set maximal depth for maximal accuracy. For this tree we get R^2-scores of:

model	fieldstrength	2nd order phase	3rd order phase	mean score
decision tree	0.986324	0.973060	0.961616	0.9737

Table 5.2: R^2-score for decision tree, no set depth limit - no noise

On the spectra without any noise, the decision tree shows good performance. The prediction to true value distribution displays this fact in a similar manner and can be found in Figure A.1. We can therefore increase the difficulty of task by adding noise onto the spectra. Calculating the R^2-scores results in:

model	fieldstrength	2nd order phase	3rd order phase	mean score
decision tree	0.950302	0.489852	0.028472	0.4895

Table 5.3: R^2-score for decision tree, no depth limit - 1% noise

While the R^2-score for fieldstrength still shows good performance, the scores for 2nd and 3rd order phase decrease significantly. The noise increases variance considerably, as can be seen in the distributions in Figure 5.3:

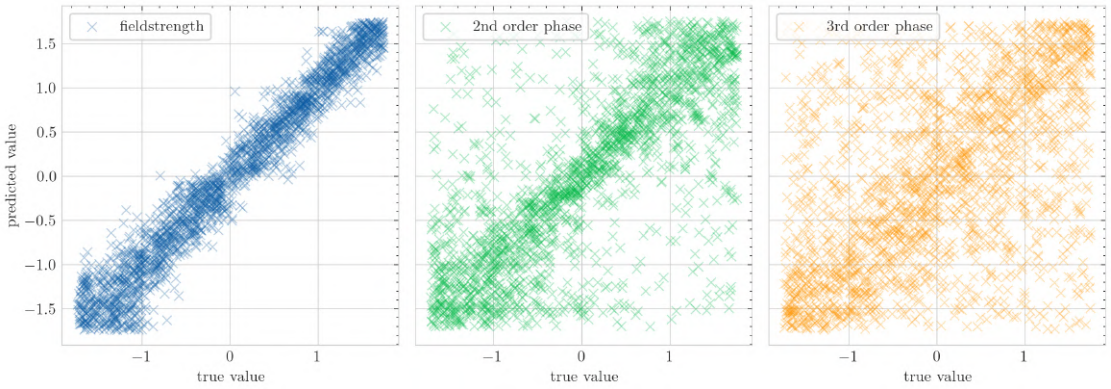


Figure 5.3: Predicted against true values for spectra with 1% noise. The fieldstrength can be predicted quite accurately, but for 2nd and especially 3rd order phase the variance increases significantly. The seemingly "square"-shaped distribution can be explained by the uniform sampling of different hyperparameters going into the simulation. The model will not extrapolate further than the most extreme training sample it has seen.

The additional noise results in substantial overfitting of the model. One can improve the model by limiting the maximal tree depth and setting a minimum sample number for leaf nodes.

model	fieldstrength	2nd order phase	3rd order phase	mean score
decision tree	0.961640	0.662934	0.353320	0.6593

Table 5.4: R^2-scores for a decision tree with a maximum depth of 8 layers and a minimum if 3 samples in the last leaf nodes. The model was trained and evaluated for the case of 1% noise.

The accuracy increases for all parameters. The distributions can be found in Figure 5.4:

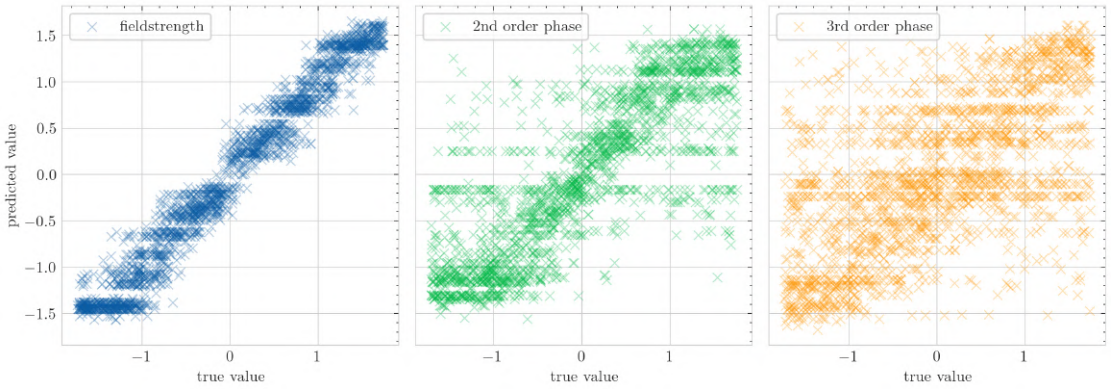


Figure 5.4: Predicted against true values for the improved decision tree trained on spectra with 1% noise. Compared to Figure 5.3 a visible improvement is noticeable. The horizontal formations can be ascribed to the decision tree being a constant function approximator. By limiting the maximal depth to 8 splits, the output variation is reduced to a maximum of $2^8 = 256$ different values, which can then be seen as discrete horizontal lines.

The decision tree works well for the generated absorption spectra without any noise. The small variations in noisy data however, lead to a significant decrease in performance due to overfitting. By pruning the decision tree, the maximum number of splits can be reduced and the tree generalizes better.

5.2.3 MLP- Multi-Layer Perceptron and variations

All deep learning models were implemented using the public python library tensorflow[18].

Base model
We start off with a feed forward dense layer neural network with two hidden layers.

layer	number of neurons	activation function
input layer	3276	linear
hidden layer 1	100	ReLU
hidden layer 2	100	ReLU
output layer	3	linear

Table 5.5: Base mlp network architecture. The total number of trainable parameters can be calculated by adding the number of weights and intercepts of each weight matrix together, e.g. $[(3276 \cdot 100) + 100] + [(100 \cdot 100) + 100] + [100 \cdot 3) + 3] = 338.103$ total parameters.

For deep learning models it can be interesting to evaluate the training progress by charting the loss against the number of completed epochs. All models were trained using the same batch size, the same number of epochs, and the same variant of ADAM optimizer with adaptive learning rate [12]. Furthermore, the same loss function; mean-squared error was chosen as well. This allows for a scientific comparison.
With the given base model as defined in Table 5.5 the model was trained and tested for each of the four noise levels (0%,1%,3%,5% noise) separately:

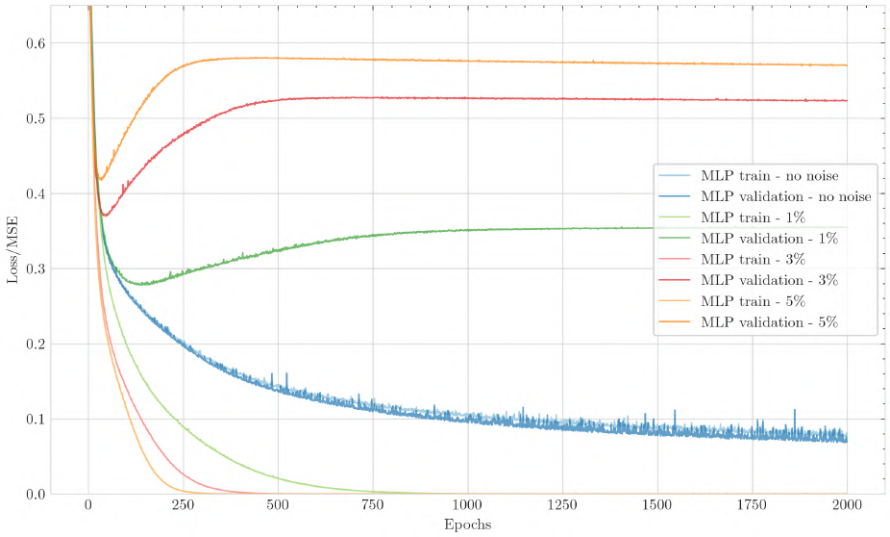


Figure 5.5: Loss plotted against epochs during training for the mlp base model. For spectra without any noise, the training and validation loss overlap and converge to nearly zero. This indicates a very good fit. For all models with noise, the training and validation loss diverge from each other after a few hundred epochs. The training error converges to exactly zero, which internally signifies that the models are memorizing all training examples, resulting in overfitting and, therefore, a worse performance on unseen validation data.

In Figure 5.5 the loss-curves during training for the base mlp model are depicted. The training on absorption spectra without any noise runs stable, converging to near zero. Training on noisy spectra, however, leads to considerable overfitting, getting worse for increased levels of noise.
For every model, the best checkpoint is found in the minimum of the validation loss. These are the checkpoints chosen to evaluate the models in the further analysis.
The trained models are then evaluated on the test data set. For that, we calculate the R^2-scores:

model	fieldstrength	2nd order phase	3rd order phase	mean score
mlp-spectrum	0.998726	0.918998	0.868945	0.928890
mlp-noise-1%	0.973458	0.709761	0.471788	0.718336
mlp-noise-3%	0.956977	0.606768	0.310213	0.624653
mlp-noise-5%	0.935732	0.546659	0.235063	0.572485

Table 5.6: R^2-score for base mlp model on different noise levels. The checkpoints to evaluate were chosen, such that the validation loss is minimized, as seen in Figure 5.5. The suffix denotes the different noise levels the model was trained and evaluated on.

With increasing noise levels, the model's performance decreases. The fieldstrength slighty loses accuracy, while the 2nd-order phase and most notably the 3rd-order phase lose significant accuracy. This can be seen in the predicted to true value distributions as well. The predicted values plotted against the true values are depicted in Figure A.2.

Small model
The most natural way of reducing overfitting, is by decreasing the number of total parameters. If a smaller network is able to perform similarly good we can reduce the training time for further experiments significantly.
Thus, the experiments were repeated for a smaller multi-layer perceptron. The network architecture chosen is listed in Table 5.7:

layer	number of neurons	activation function
input layer	3276	linear
hidden layer 1	10	ReLU
hidden layer 2	10	ReLU
output layer	3	linear

Table 5.7: Small mlp architecture - This network has a total of 32.913 trainable parameters. A larger network was not inspected due to multiplicative scaling of parameters. Assuming 1000 neurons in the first hidden layer, the model would already have over three million parameters with a size of several hundred MB.

For the comparatively small mlp, the loss graphs can be inspected as well:

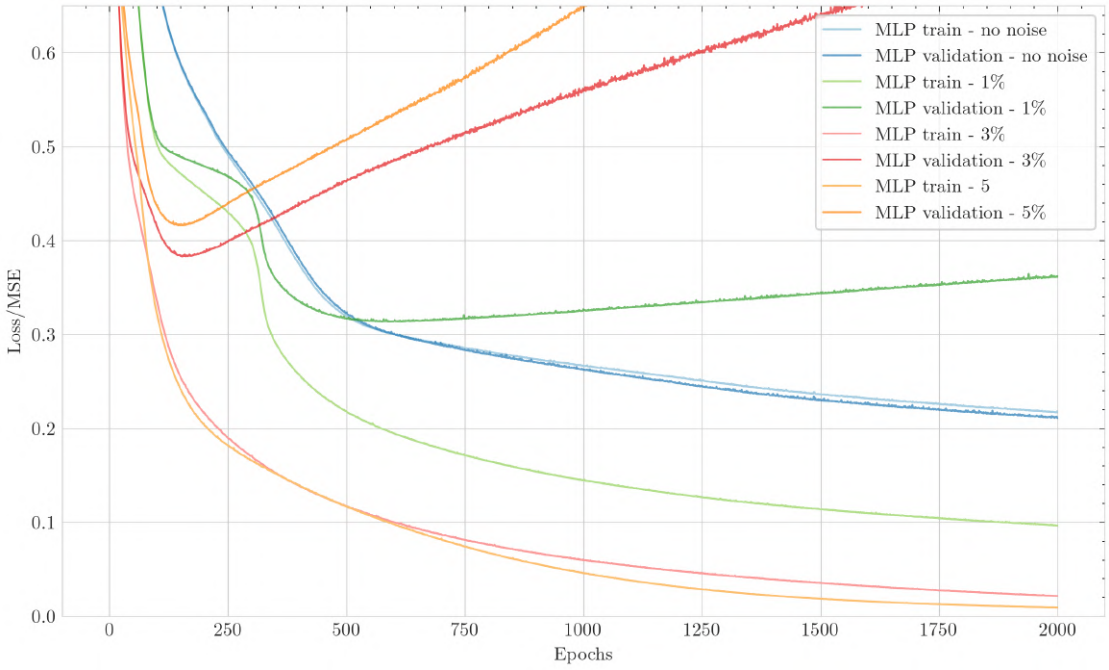


Figure 5.6: Loss plotted against epochs during training for the small mlp model. For spectra without any noise, the training and validation loss overlap and converge, but the convergence is worse than for the larger model. Therefore, the model is underfitting on noiseless spectra. For all models with noise, the training and validation loss diverge from each other, here even steeper than in the model before. The training errors slowly converge to exactly zero. For every model the best checkpoint is found in the minimum of the validation loss.

The loss-curves give insight into the training of the smaller mlp. The loss for training on spectra without noise is not clearly converged, however, it is still approximately twice as high as for the model larger before. Unlike for the first model, the models trained on noisy data converge much slower, or for the higher levels of noise not at all. The performance overall is much worse compared to the larger model from before. The R^2-score resemble similar results:

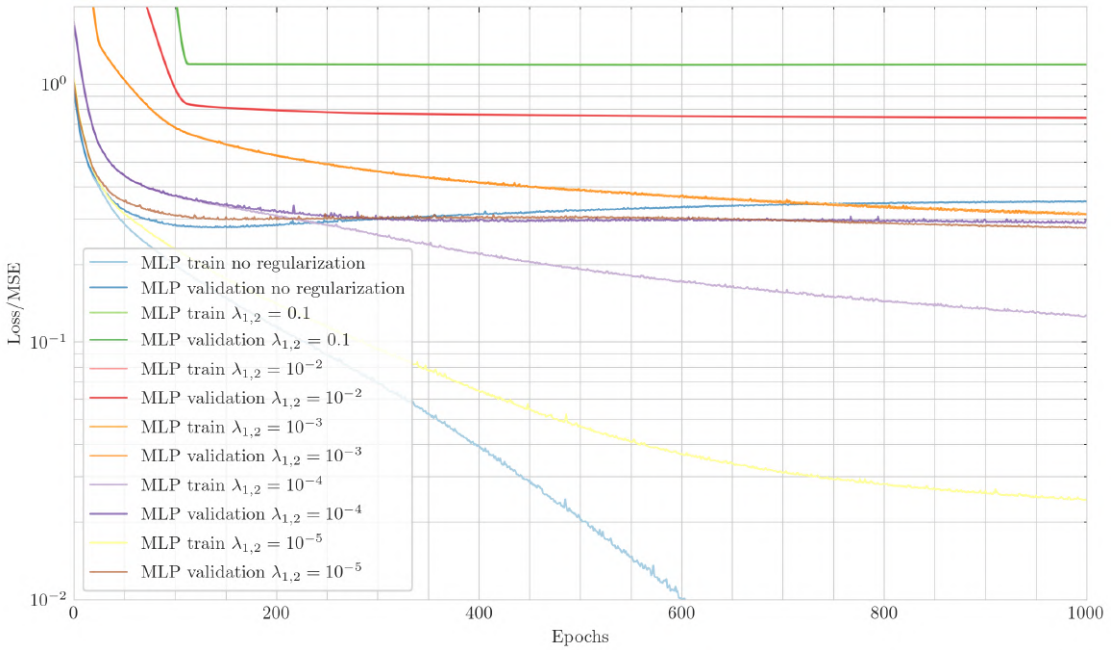
parameter	fieldstrength	2nd order phase	3rd order phase	mean score
mlp-small-spectrum	0.950677	0.756723	0.642706	0.783369
mlp-small-noise-1%	0.975888	0.630462	0.450309	0.685553
mlp-small-noise-3%	0.934141	0.602394	0.297975	0.611503
mlp-small-noise-5%	0.924072	0.563778	0.217278	0.568376

Table 5.8: R^2-scores for the smaller mlp. Compared to the base mlp model, it performs worse for all noise levels.

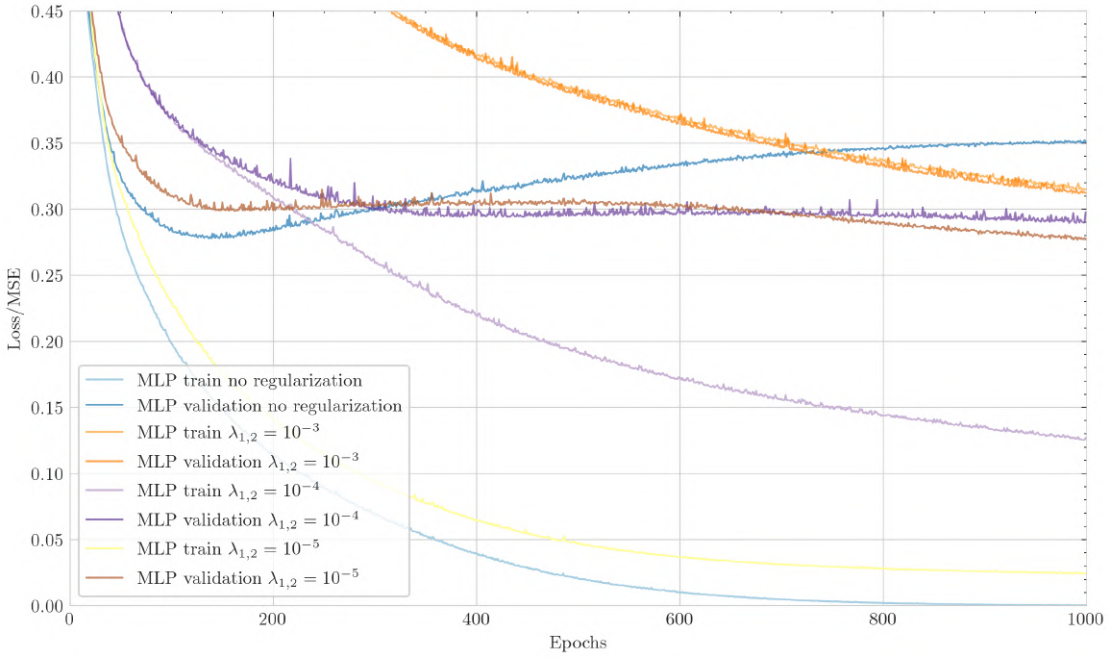
In a next step one can try to improve the models by implementing regularization as described in section three.

Regularized model

Since the larger model is the better model of the two, we try to further improve its performance by reducing overfitting. Through the addition of regularization terms to the loss function, the loss-landscape is smoothed and the model should generalize better. Since it is hard to find the best scaling parameters λ_1 for L_1-regularization and λ_2 for L_2-regularization, we rather focus on evaluating the impact of different magnitudes for λ_1 and λ_2. Both regularization terms were used and equally scaled with magnitudes ranging from $\{0.1, 0.01, \ldots, 10^{-5}\}$.
Lastly, we restrict us to only evaluate the spectra with one percent noise, since overfitting occurred only in noisy spectra.



(a) The loss over epochs for different magnitudes of regularization. Since the scale spans multiple magnitudes, a log scale is used for better visual representation. For large parameters $\lambda_{1,2}$ the additional loss outweighs the actual error during training. The model only learns to minimize the network weights.



(b) The same figure, excluding large loss values and using the normal scale.

Figure 5.7: Loss curves of base mlp with L_1 and L_2 regularization. For too large hyperparameters $\lambda_{1,2}$, the regularization loss predominates, and the actual error has no impact on the total loss. For smaller magnitudes of $\lambda_{1,2}$ the validation loss improves, while the minimal-loss model improves only slightly.

Compared to the model without regularization, the models are not able to memorize the training dataset and therefore, the divisions between validation and training error are reduced .To evaluate the models we calculate the R^2-scores to gain further insight:

model	fieldstrength	2nd order phase	3rd order phase	mean score
mlp-kernelreg-$\lambda_{1,2} = 10^{-1}$	-0.000048	-0.000122	-0.000042	-0.000071
mlp-kernelreg-$\lambda_{1,2} = 10^{-2}$	0.968848	0.001596	0.001788	0.324077
mlp-kernelreg-$\lambda_{1,2} = 10^{-3}$	0.988420	0.753394	0.640261	0.794025
mlp-kernelreg-$\lambda_{1,2} = 10^{-4}$	0.996004	0.806470	0.613220	0.805231
mlp-kernelreg-$\lambda_{1,2} = 10^{-5}$	0.995711	0.738964	0.453717	0.729464

Table 5.9: R^2-scores for different regularization parameters λ_1 and λ_2. For regularization parameters $\lambda_{1,2} = 10^{-1}$, the regularization error outweighs the actual loss and the network only learns to minimize all network weights. For the second magnitude $\lambda_{1,2} = 10^{-2}$, only the fieldstrength has enough impact to affect the loss minimization, 2nd- and 3rd-order phase are ignored during the learning process. For the last three magnitudes, the total score improves with declining regularization parameters $\lambda_{1,2}$, before the score begins to fall in all three target categories for too small $\lambda_{1,2}$'s.

The R^2-scores suggest, the best regularization parameter to be somewhere around $(10^{-3} - 10^{-5})$,

though particular targets seem to be affected differently. Furthermore Figure 5.7b suggests not all models are fully converged. Further analysis is necessary to investigate this surmise.

Dropout model
For this variation, the base mlp was modified using dropout layers as previously described in section 3.3.3. . Each node of the two hidden layers has a certain dropout percentage during training. Three different probabilities were chosen; 10%, 30% and 50%. Similar to regularization the same dropout probability was chosen for both layers.
Figure 5.8 depicts the training progress for these models.

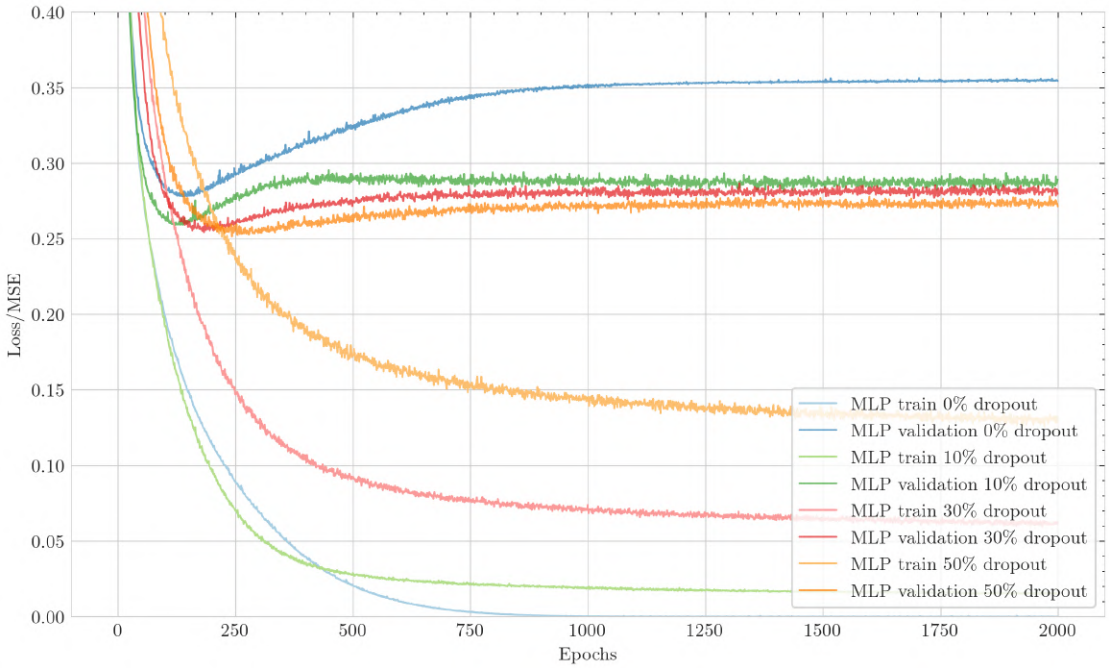


Figure 5.8: Loss graphed against epochs during training for the base mlp model with different dropout rates inbetween the hidden and last layers. The models were trained and evaluated on spectra with 1% gaussian noise. The impact of dropout is clearly noticeable. On the one hand the validation error converges flatter against a lower final error, on the other hand the training error significantly increases, the model is no longer capable to memorize all of the training data perfectly. The amount of dropout percentage does not seem to have a significant impact on the best performing checkpoint.

A clear improvement is visible. While the different dropout rates seem to make no difference for the validation loss, the training error raises with increasing dropout rate. The improved performance is resembled in the R^2-scores as well:

model	fieldstrength	2nd order phase	3rd order phase	mean score
mlp-dropout-10%	0.982848	0.735941	0.490267	0.736352
mlp-dropout-30%	0.979719	0.738185	0.513958	0.743954
mlp-dropout-50%	0.974681	0.736187	0.521784	0.744217

Table 5.10: R^2-scores for different dropout probabilities. All three models seem to perform equally within statistical significance. For each target category, a different model is slightly better than the other two, however, this may also be a purely statistical variance.

5.2.4 CNN - Convolutional neural network and variations

A significant disadvantage of fully connected layers is the influence of distant input nodes onto the same output node. For all nodes in a layer, every input node is included for calculating every output. But for a peak or attenuation of intensity in the absorption spectrum, it seems plausible that only the nearby nodes hold most of the relevant information. A convolutional neural network takes advantage of local patterns and hence seems to be better suited for this case of problem.

Base CNN - convolutional neural network
The architecture of the first implemented convolutional neural network (cnn) mainly consists of convolutional layers acting in a downsampling manner and two fully connected dense layers at the end. The exact architecture is listed in Table 5.11:

layer	filters	kernel size	strides	padding	activation function	output shape
input layer	-	-	-	-	linear	3276×1
conv layer 1	8	3	3	valid	ReLU	1092×8
conv layer 2	16	3	3	valid	ReLU	364×16
conv layer 3	32	3	3	valid	ReLU	121×32
conv layer 4	64	4	4	valid	ReLU	30×64
conv layer 5	128	5	5	valid	ReLU	6×128
flatten	-	-	-	-	-	768
dense layer	-	-	-	-	ReLU	100
output layer	-	-	-	-	linear	3

Table 5.11: The network architecture of the convolutional neural network. Parameters noted in the column are explained in section 3.3.4..To reduce the size of the data passing through the network, comparatively large strides are used. It is sensible to choose a kernel size larger or equal to the number of strides, since otherwise inputs may be skipped. The output shape can be interpreted as (number of neurons $\times$ channels). Since convolutional layer only transform the input data, dense layers at the end are needed to interpret the *filtered* output. By stacking multiple convolutional layers together, the receptive field of the entire block can be increased significantly. This architecture results in a total of 128.547 trainable parameters.

This model was trained on all different noise levels as well:

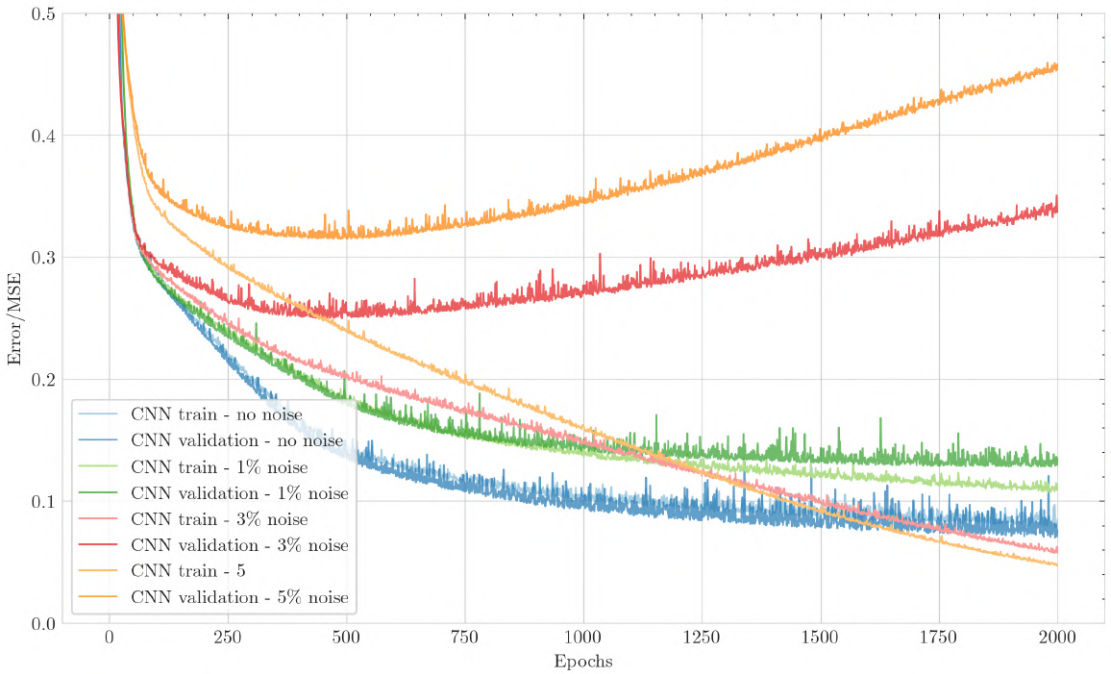


Figure 5.9: Loss plotted against epochs during training for the base cnn model with different noise levels. Compared to Figure 5.5, the model starts to overfit significantly later, or in the case of 1% noise does not overfit at all. For higher levels of noise, the model starts to overfit as well, but significantly slower than the base mlp model before.

The loss-curves of this training process, depicted in Figure 5.9 indicate a good performance. The loss for spectra without and 1% noise converges, while for higher noise levels the model begins to overfit as well. For noise levels of 3% and 5% the training error is not fully converged, however, this is irrelevant, since the validation error begins to get worse much earlier.
The R^2-scores can be calculated again, mirroring the good fits as indicated by the loss graphs:

model	fieldstrength	2nd order phase	3rd order phase	mean score
cnn-spectrum	0.999180	0.916940	0.867287	0.927802
cnn-noise-1%	0.997274	0.855261	0.731325	0.861286
cnn-noise-3%	0.989070	0.736995	0.529231	0.751765
cnn-noise-5%	0.985159	0.663978	0.398179	0.682439

Table 5.12: R^2-scores of deep learing models in comparison. For the pure spectrum without any noise, the model scores similary to the base mlp. For models with noise however, the CNN performs substantially better than all models before, outperforming every variant of the base MLP aswell.

The R^2-scores for the reconstruction of fieldstrength decrease slightly, although it has to be noted, that the actual performance difference between .99 and .9 is considerably larger than the difference in score. For 2nd- and 3rd-order phase reconstruction the score decreases significantly with increasing noise levels. This model can be improved, by expanding the number of filters

and layers.

Deep CNN
A deep neural network is a feed forward neural network with many hidden layers. The base model cnn could already qualify as a deep convolutional neural network (DCNN), but in this paragraph an improved, deeper version of convolutional neural network is implemented, the nomenclature being deep convolutional neural network (DCNN). While the last CNN mainly consists of convolutional layers in order to downsample the spectrum, this model has more convolutional layers with smaller strides and more parameters in total.

layer	filters	kernel size	strides	padding	activation function	output shape
input layer	-	-	-	-	linear	3276×1
conv layer 1	4	4	2	valid	ReLU	1637×4
conv layer 2	8	4	2	valid	ReLU	817×8
conv layer 3	8	5	1	valid	ReLU	813×8
conv layer 4	16	4	4	valid	ReLU	203×16
conv layer 5	16	5	1	valid	ReLU	199×16
conv layer 6	64	4	4	valid	ReLU	49×64
conv layer 7	64	5	1	valid	ReLU	45×64
conv layer 8	128	4	4	valid	ReLU	11×128
conv layer 9	128	3	1	valid	ReLU	9×128
conv layer 10	128	3	1	valid	ReLU	7×128
flatten	-	-	-	-	-	896
dense layer	-	-	-	-	ReLU	100
output layer	-	-	-	-	linear	3

Table 5.13: The network architecture of the deep convolutional neural network. With 10 convolutional layers and a total of 248.471 trainable parameters, this dcnn is about twice as large as the cnn model before.

To counteract the vanishing gradient in earlier layers, this model was trained for 5000 epochs. Since there are much more layers it took significantly longer to train. The training progress is visualized in Figure 5.10.

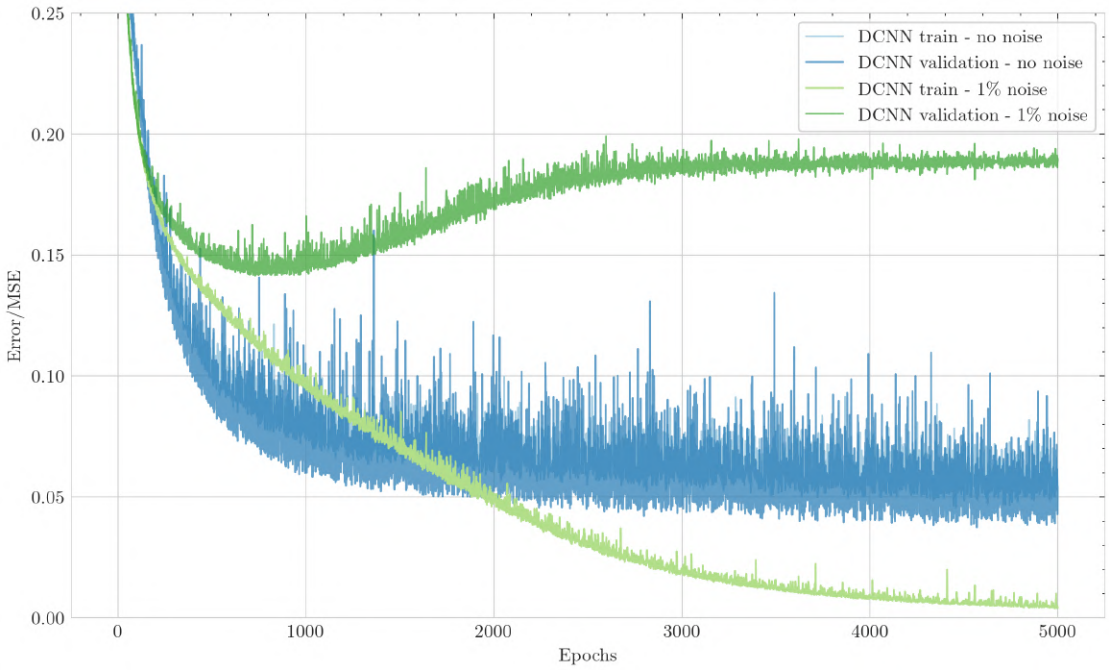


Figure 5.10: Loss plotted against epochs during training for the DCNN model for 1% and no noise on spectra. For spectra without noise, the model performs astonishingly well, although the entire training seems to be less stable, which is indicating a sharper, rougher loss landscape. When adding 1% noise, the model starts to overfit, a possible reason being the significantly larger amount of parameters compared to the CNN.

The R^2-scores can be calculated as well:

model	fieldstrength	2nd order phase	3rd order phase	mean score
dcnn-spectrum	0.999739	0.954872	0.931922	0.962178
dcnn-noise-1%	0.995842	0.819356	0.739126	0.851441

Table 5.14: R^2-scores of the deep convolutional neural network. For spectra without any noise the model fully converges, while for spectras with 1% noise the model overfits. Since the model already starts to overfit for the lowest level of noise and takes approximately 5 times longer to train than the smaller cnn, the analysis was stopped here and not expanded further onto all noise levels.

Compared to the base cnn the dcnn model performs better on the spectra without any noise. On spectra with 1% noise the dcnn overfits very early resulting in a similar performance compared to the cnn.

5.2.5 Summary

It is sensible to inspect the fundamental models and deep learning models separately, since the latter models had 1600 less samples during training. For the simple regression models we get:

dataset	model	fieldstrength	2nd order phase	3rd order phase	mean score
no noise	linear regression	0.986690	-0.209403	-0.403350	0.1246
	decision tree - no depth limit	0.986324	0.973060	0.961616	0.9737
1% noise	decision tree - no depth limit	0.950302	0.489852	0.028472	0.4895
	decision tree - max depth = 8, min leaf samples = 3	0.961640	0.662934	0.353320	0.6593

Table 5.15: R^2-scores of fundamental machine learning models in comparison. Highlighted green the best model for each data set is marked.

For spectra without noise, the fieldstrength can be reconstructed by both fundamental models. The spectral phase terms, however, can only be reconstructed by the decision tree, while the linear regression model performs much worse. For spectra with 1% noise the decision tree with no set branch limit begins to overfit, resulting in a reduction in performance. By limiting the number of splits and setting a minimal number of samples in the end nodes, the performance can be increased slightly.
The R^2-score performance of deep-learning models is summarized in Table 5.16.

dataset	model	fieldstrength	2nd order phase	3rd order phase	mean score
no noise	mlp	0.998726	0.918998	0.868945	0.928890
	mlp-small	0.950677	0.756723	0.642706	0.783369
	cnn	0.999180	0.916940	0.867287	0.927802
	dcnn	0.999739	0.954872	0.931922	0.962178
1% noise	mlp	0.973458	0.709761	0.471788	0.718336
	mlp-small	0.975888	0.630462	0.450309	0.685553
	mlp-kernelreg-$\lambda_{1,2} = 10^{-1}$	-0.000048	-0.000122	-0.000042	-0.000071
	mlp-kernelreg-$\lambda_{1,2} = 10^{-2}$	0.968848	0.001596	0.001788	0.324077
	mlp-kernelreg-$\lambda_{1,2} = 10^{-3}$	0.988420	0.753394	0.640261	0.794025
	mlp-kernelreg-$\lambda_{1,2} = 10^{-4}$	0.996004	0.806470	0.613220	0.805231
	mlp-kernelreg-$\lambda_{1,2} = 10^{-5}$	0.995711	0.738964	0.453717	0.729464
	mlp-dropout-10%	0.982848	0.735941	0.490267	0.736352
	mlp-dropout-30%	0.979719	0.738185	0.513958	0.743954
	mlp-dropout-50%	0.974681	0.736187	0.521784	0.744217
	cnn	0.997274	0.855261	0.731325	0.861286
	dcnn	0.995842	0.819356	0.739126	0.851441
3% noise	mlp	0.956977	0.606768	0.310213	0.624653
	mlp-small	0.934141	0.602394	0.297975	0.611503
	cnn	0.989070	0.736995	0.529231	0.751765
5% noise	mlp	0.935732	0.546659	0.235063	0.572485
	mlp-small	0.924072	0.563778	0.217278	0.568376
	cnn	0.985159	0.663978	0.398179	0.682439

Table 5.16: R^2-scores of deep learning models in comparison. The convolutional neural network outperforms the mlp and all of its variations. Highlighted in green are the best performing models evaluated by their mean score.

For absorption spectra without noise, the dcnn model performs best overall, as well as in every target category. Notably, the fieldstrength can be reconstructed best in all models.
For spectra with 1% noise a variety of experiments was run. Different regularization methods

were implemented and tested. Among the regularization techniques, the dropout-method performs more stable, while the L_1- and L_2-regularization methods perform best. Variations in the magnitude of the regularization parameters λ_1 and λ_2 produce large differences in the resulting models.
Overall, however, the cnn closely followed by the dcnn performs best for this task.
The higher levels of noise depict similar results, the cnn significantly outperforms the other two mlp models.

6 Conclusion

The goal of this thesis was to show the viability of machine learning techniques for the analysis of absorption spectra, by reconstructing the laser pulse properties which generated the absorption spectra. The toy problem tackled, was the reconstruction of fieldstrength, 2nd- and 3rd-order phase of a laser pulse interacting with a thin medium from an measured absorption spectrum. The performance of various machine-learning models on this task has been investigated. Additionally, the influence of noisy data on the different model's performance has been investigated.
In a first step 100.000 spectra were numerically generated using a few-level model. A laser pulse with a frequency of $190[\frac{1}{\Delta t}]$ and spectral width of $30[\frac{1}{\Delta t}]$ was chosen and the fieldstrength and phase parameters of the pulse were varied in a sensible manner. The medium was modeled to be an arbitrary twenty-level system, where all nineteen excited states couple to the ground state but not to each other. The exact parameters can be found in section (4.1) table (4.1) and (4.2). With 10.000 of those generated spectra, different models were trained and evaluated. To increase the difficulty of the task three different noise levels were added to the spectra and new models were trained on and tested on those.
The decision tree proved to be the best model for spectra without noise overall, for the fieldstrength specifically however, the deep convolutional neural network performs best[2]. A combination of multiple models, in principle, could yield even better performance, the investigation of which, however, is beyond the scope of this thesis.
For absorption spectra with additional gaussian noise, the decision tree started to overfit rapidly, hence more stable, complex models were chosen for further analysis. Here, the comparatively smaller convolutional neural network performed best overall, while the larger DCNN started overfitting, even for very small levels of noise.

Outlook

The results indicate encouraging prospects for machine-learning models in absorption spectroscopy, or more generally for solving inverse, non-analytically solvable problems. Another promising, recently emerged neural network architecture, specifically designed for inverse problems, is the so-called invertible neural network (INN) [19]. However, due to time constraints and different sampling methods during training, an entirely different work environment would have been needed to train and compare this kind of model's performance on this toy problem.
Further improvements not discussed or implemented could be achieved by using more of the available data, though better hardware would be necessary for training in that case. For datasets with noise, data augmentation [20] should counteract overfitting as well.
These improvements were not implemented, to ensure comparability between models and thus different other techniques were explored instead.

[2]Though it should be noted that all deep learning models had 1600 less training samples, since those were used to validate the training performance

A Appendix

A.1 Decision Tree

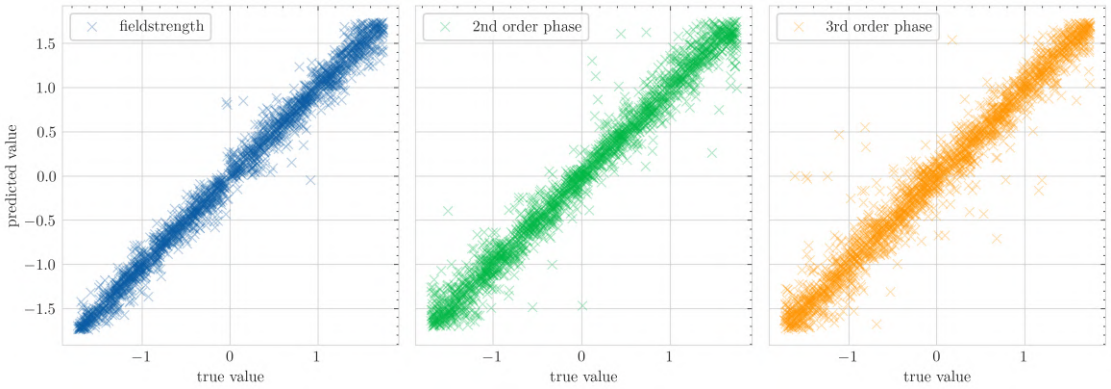
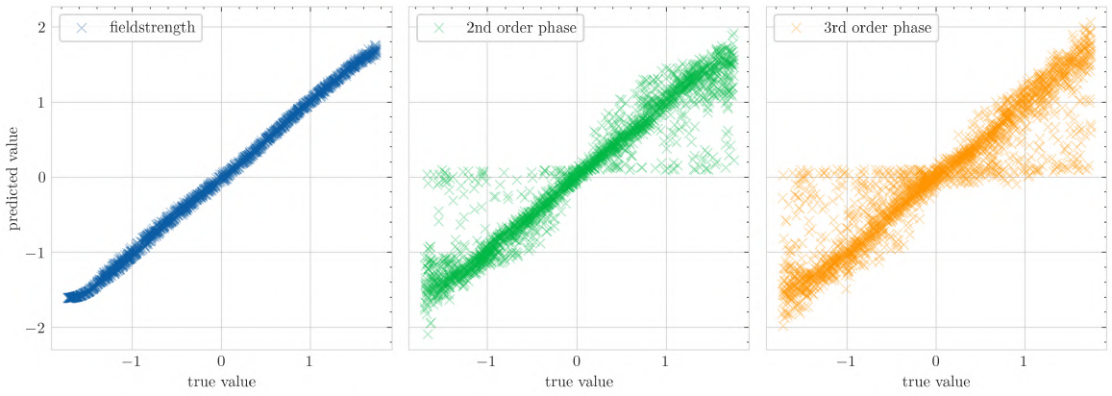
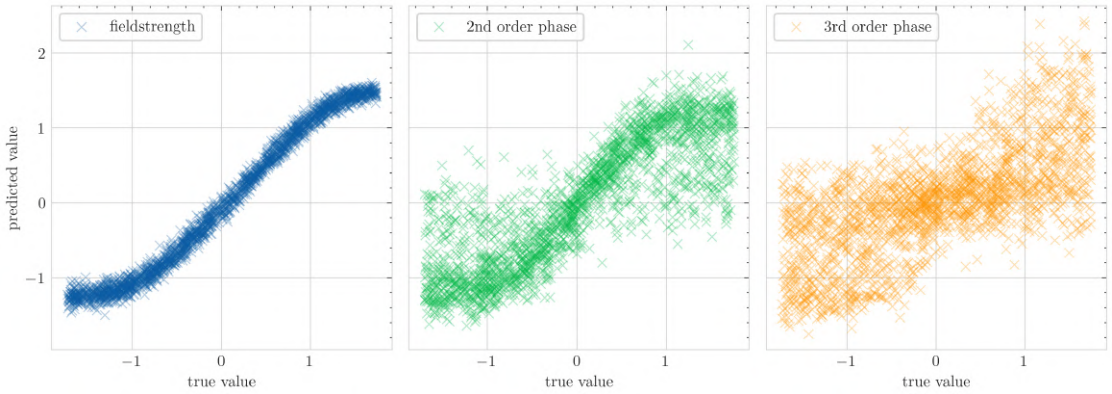


Figure A.1: Predicted values against targets for an unlimited branched decision tree - no noise. We can see very little spread in the graphs, resembling the high R^2-scores in table (5.2).

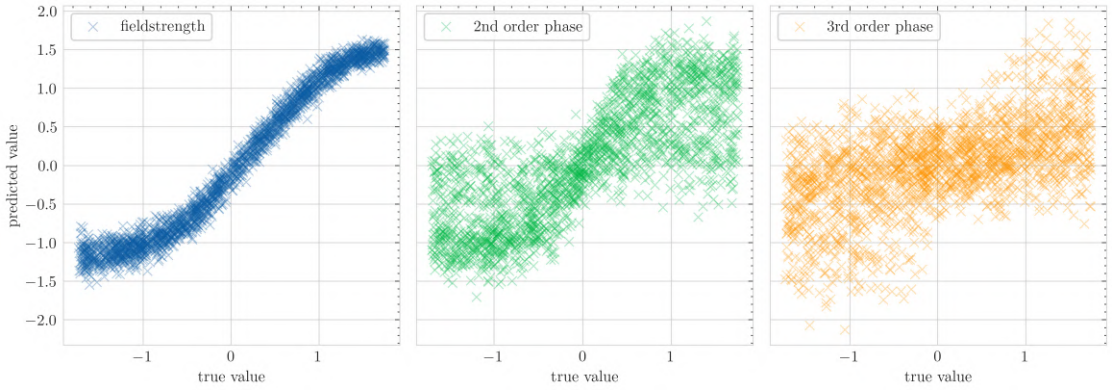
A.2 MLP



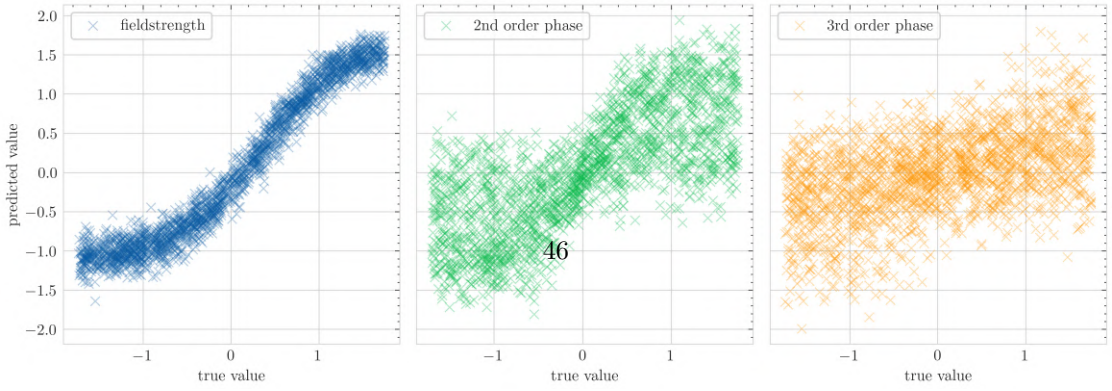
(a) model trained on spectra without any noise



(b) model trained on spectra with 1% noise



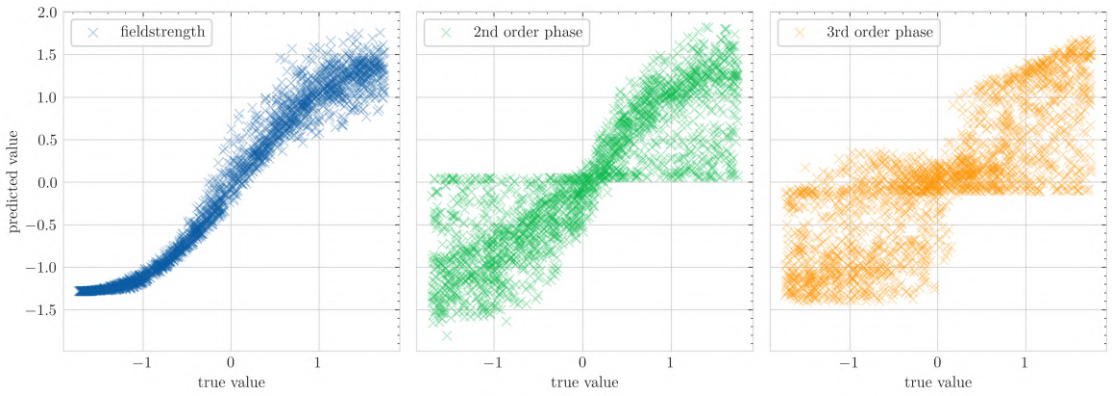
(c) model trained on spectra with 3% noise



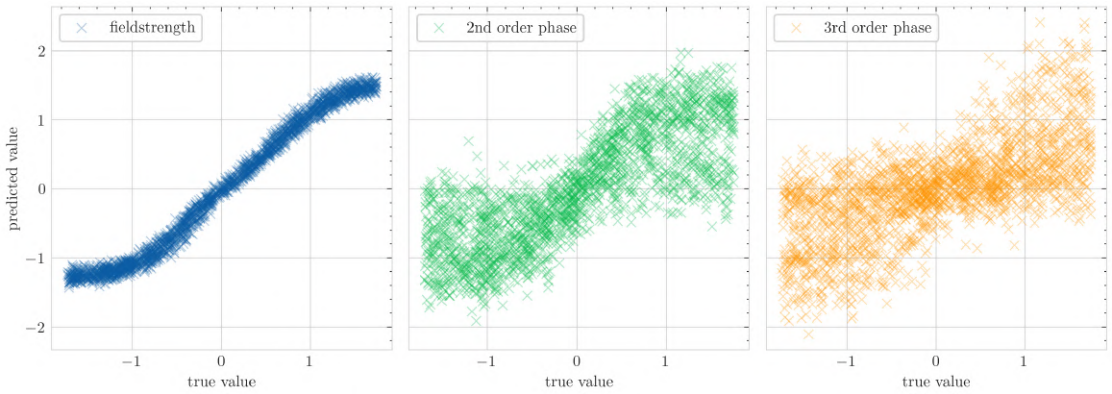
(d) model trained on spectra with 5% noise

Figure A.2: Predicted values graphed against true values for base mlp constructed as described in table (5.5). In **(a)-(d)** are the results for different noise levels depicted.

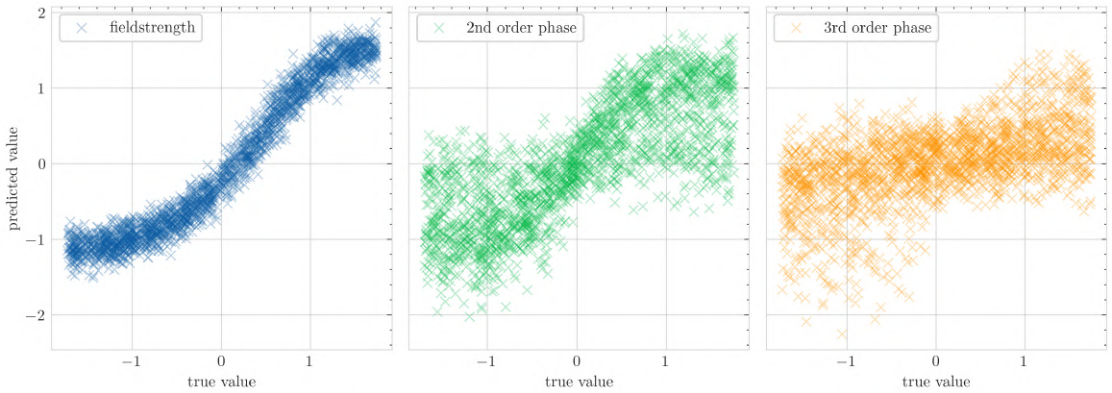
A.2.1 MLP small



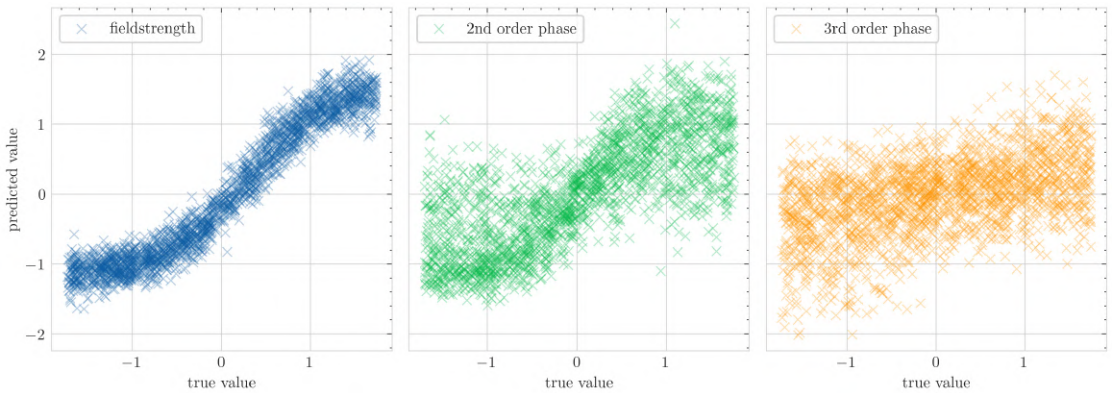
(a) model trained on spectra without any noise



(b) model trained on spectra with 1% noise



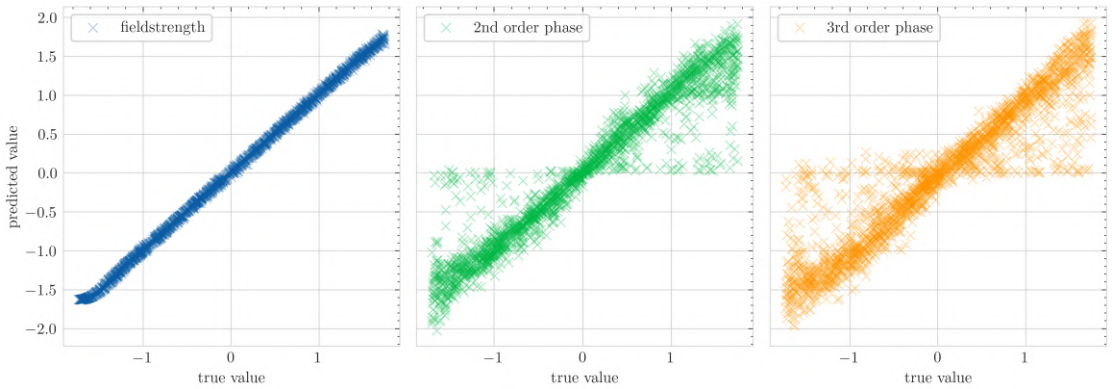
(c) model trained on spectra with 3% noise



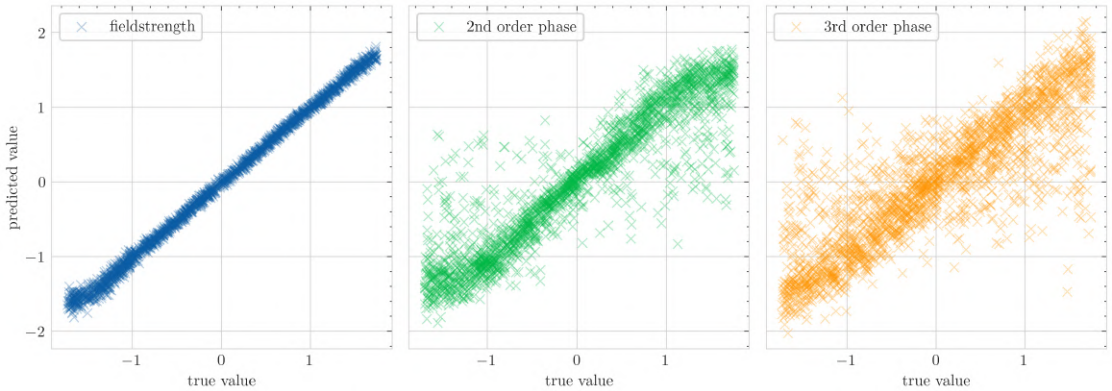
(d) model trained on spectra with 5% noise

Figure A.3: Predicted values graphed against true values for the small mlp constructed as described in table (5.7). In **(a)-(d)** are the results for different noise levels depicted.

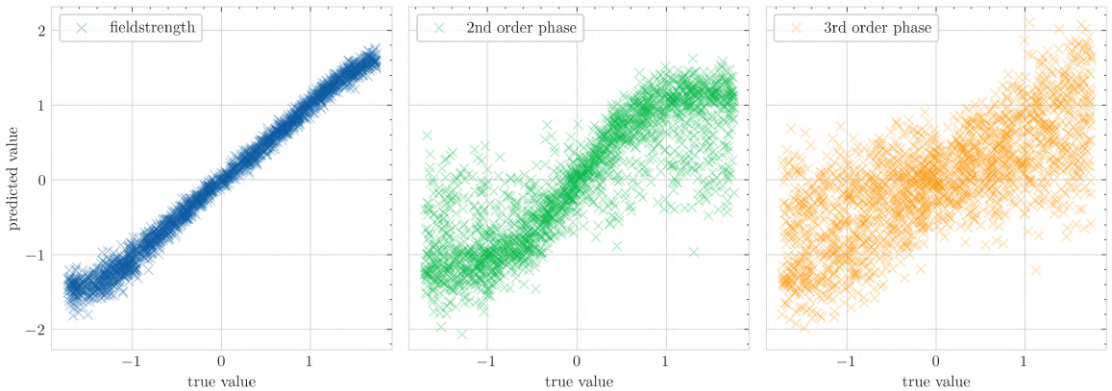
A.3 CNN



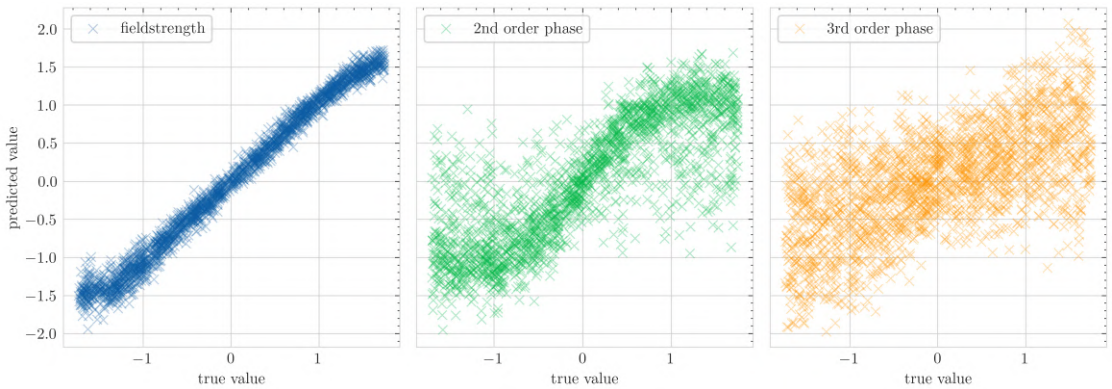
(a) model trained on spectra without any noise



(b) model trained on spectra with 1% noise



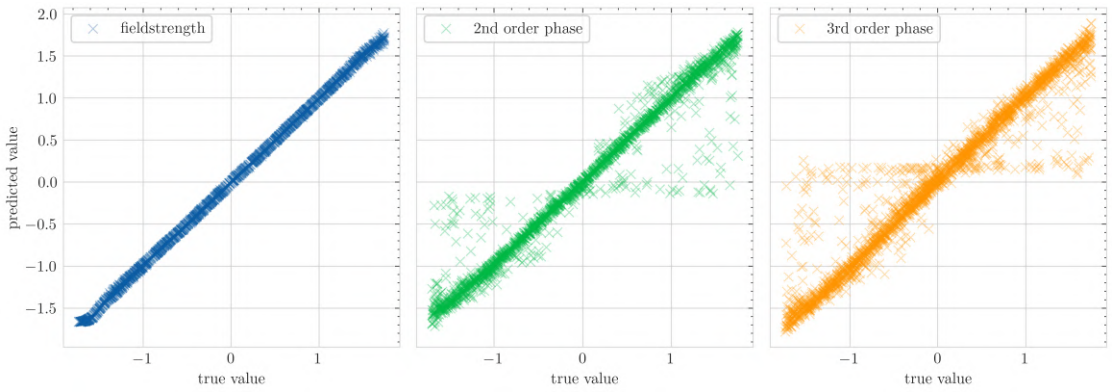
(c) model trained on spectra with 3% noise



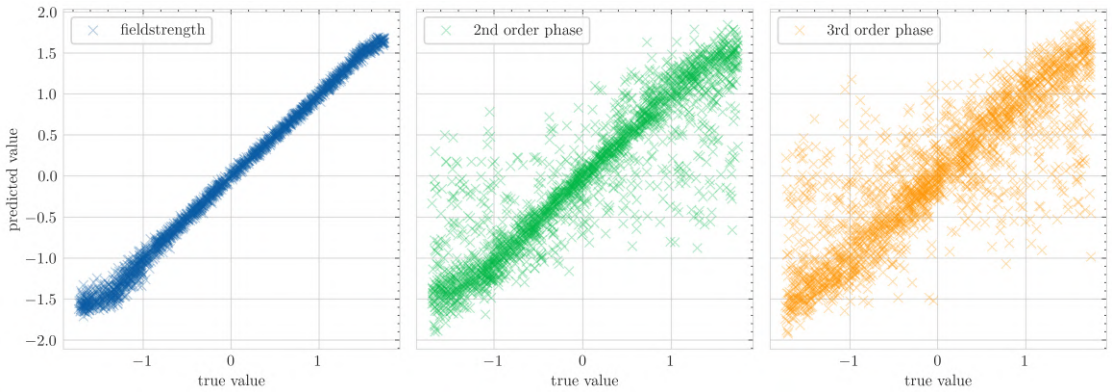
(d) model trained on spectra with 5% noise

Figure A.4: Predicted values graphed against true values for base CNN constructed as described in table (5.11). In **(a)-(d)** are the results for different noise levels depicted.

A.3.1 DCNN



(a) model trained on spectra without any noise



(b) model trained on spectra with 1% noise

Figure A.5: Predicted values graphed against true values for DCNN constructed as described in table (5.13). In **(a)-(b)** are the results for the two different noise levels depicted.

References

[1] W. McCulloch, "y pitts, w (1943), a logical calculus of the ideas immanent in nervous activity," *Bull. of Math. Biophysics*, vol. 5, p. 116.

[2] J. Diels, W. Rudolph, P. Liao, and P. Kelley, *Ultrashort Laser Pulse Phenomena*. Optics and photonics, Elsevier Science, 2006.

[3] C. R. Ott, "Attosecond multidimensional interferometry of single and two correlated electrons in atoms," 2012.

[4] J. C. Baggesen, E. Lindroth, and L. B. Madsen, "Theory of attosecond absorption spectroscopy in krypton," *Phys. Rev. A*, vol. 85, p. 013415, Jan 2012.

[5] M. D. Hartmann, "Attosecond dynamics of strong-field generated ions," 2021.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] M. Hafeez, M. Rashid, H. Tariq, Z. Abideen, S. Alotaibi, and M. Sinky, "Performance improvement of decision tree: A robust classifier using tabu search algorithm," *Applied Sciences*, vol. 11, p. 6728, 07 2021.

[8] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Annals of Data Science*, vol. 9, 04 2022.

[9] H. B. Curry, "The method of steepest descent for non-linear minimization problems," *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261, 1944.

[10] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[11] A. Griewank, "On automatic differentiation," pp. 83–108, 1989. MCS-P10-1088, 1989. Appears in Mathematical Programming: Recent Developments and Applications, eds. M. Iri, K. Tanabe.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[13] T. P. Trappenberg, "183Cyclic models and recurrent neural networks," in *Fundamentals of Machine Learning*, Oxford University Press, 11 2019.

[14] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," 2017.

[15] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[16] F. Mosteller and J. W. Tukey, "Data analysis, including statistics," in *Handbook of Social Psychology, Vol. 2* (G. Lindzey and E. Aronson, eds.), Addison-Wesley, 1968.

[17] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation," *PeerJ Computer Science*, vol. 7, p. e623, 2021.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[19] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks," 2018.

[20] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

Danksagung

Mein Dank gilt insbesondere:

Prof. Dr. Thomas Pfeifer für die Möglichkeit, diese Arbeit in seiner Abteilung zu absolvieren, sowie der herausragenden Betreuung.

apl. Prof. Dr. Jörg Evers für die Übernahme der Zweitkorrektur.

Dr. Christian Ott für die Aufnahme in seine Arbeitsgruppe, sowie zahlreichen Vorschlägen und Erklärungen.

Dr. Marc Rebholz und **Dr. Maximilian Hartmann** für die hervorragende Betreuung und Unterstützung beim Erstellen dieser Arbeit.

Maximilian Richter für die Bantwortung vieler Fragen und Feedbacks.

Dr. Marc Rebholz und **Alexander Magnolia** für das Korrekturlesen und die Verbesserungsvorschläge.

Meiner Familie und Freunden

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 11.09.2022,