

Hochschule Darmstadt

– Fachbereiche Mathematik und
Naturwissenschaften & Informatik –

Kolmogorov-Arnold-Networks For Symbolic Regression In Fracture Mechanics

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

im Studiengang Data Science

vorgelegt von

Mats Hendrik Nicolai

Matrikelnummer: 1128179

Referent : Prof. Dr. Felix Lenders

Korreferent : Prof. Dr. Guillermo Requena

Ausgabedatum : 11.11.2025

Abgabedatum : 18.05.2026

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 18.05.2026

Mats Hendrik Nicolai

ABSTRACT

This master thesis explores Kolmogorov-Arnold networks (KANs), a new type of neural network (NN), and their use case for symbolic recovery. The focus lies on reconstructing parts of the Williams series.

The data for the experiments was generated to simulate the first terms of the Williams series. Both the behaviour close to singularities different techniques to fixate analytical dunctions to edges of the network were investigated. Furthermore, the first term of the series was successfully recovered.

To increase efficiency in training, systematic hyperparameter optimization was conducted, where regularization and architecture were examined.

Additionally, the measure used for selecting functions on the edges was investigated. It was found that that assigning complexity values to analytical functions guides edge assignment has a positive effects on training and reconstruction of the underlying formula.

Overall, KANs are suited for symbolic recovery. However, they do behave sensitively with regards to hyperparameters. As a result further research is necessary to achieve optimal results using KANs for symbolic regression tasks.

ZUSAMMENFASSUNG

Diese Masterarbeit befasst sich mit der symbolischen Regression von Termen der Williamsreihe mit Hilfe von Kolmogorov-Arnold Netzwerken, einer neuen Art neuronaler Netze. Dabei wird untersucht, inwiefern sich KANs für die symbolische Regression eignen. Getestet wird, ob es möglich ist, Teile der Williamsreihe symbolisch wiederzufinden.

Die Daten für das Training der Modelle wurden auf Basis der ersten Terme der Williamsreihe simuliert. Dabei wurde zunächst sowohl das Verhalten in Singularitätsnähe untersucht, als auch Unterschiede in Herangehensweisen an die symbolische Fixierung herausgearbeitet. Anschließend wurden erfolgreich Tests am ersten Term der Williamsreihe durchgeführt.

Um das Trainieren der Modelle effizienter und strukturierter zu gestalten, wurde eine systematische Hyperparameteroptimierung durchgeführt, bei der Regularisierung und Netzwerkarchitektur untersucht wurden.

Zusätzlich wurde die Auswahl der Funktionen untersucht, die bei der symbolischen Wiederherstellung auf den einzelnen Kanten zum Einsatz kommen. Dabei stellte sich heraus, dass ein Wert, der einzelnen analytischen Funktionen bestimmte Komplexitäten zuweist, sich positiv auf das Training und die Rekonstruktion der gesuchten Formeln auswirkt.

Die Ergebnisse zeigen, dass Kolmogorov-Arnold Netzwerke gut geeignet sind, um den Daten zugrunde liegende Funktionen zu rekonstruieren. Allerdings reagieren die Modelle sensibel auf die Wahl der Hyperparameter. Die Arbeit verdeutlicht somit, dass Potenzial in Kolmogorov-Arnold Netzwerken zur Nutzung für symbolische Regression steckt, wenngleich weitere Untersuchungen erforderlich sind, um Stabilität und Optimierung der Modelle gewährleisten zu können.

CONTENTS

I Thesis

1	Introduction	2
1.1	Background and Context	2
1.2	Research Questions	2
1.3	Structure of the Thesis	3
2	Theoretical Background	4
2.1	Linear Elastic Fracture Mechanics	4
2.2	Crack Tip Field and Williams Series	5
2.2.1	Williams Series	5
2.3	Machine Learning	6
2.3.1	Supervised Learning	6
2.3.2	Multi-layer Perceptrons	8
2.3.3	Regularization	12
2.4	Symbolic Regression	12
2.4.1	Symbolic Regression Methods	14
2.5	Kolmogorov-Arnold Network	15
2.5.1	KAN Architecture	15
2.5.2	Regularization	20
3	Methods	22
3.1	Data Generation and Preparation	22
3.1.1	Data Generation with pykan-package	22
3.1.2	Creating Datasets for Synthesized Data	23
3.2	KAN	24
3.2.1	Initializing a KAN model	24
3.2.2	Training a KAN model	25
3.2.3	Symbolic Recovery	25
3.2.4	Pruning	26
3.2.5	Plotting Model Structure	26
3.2.6	Complexity of Symbolic Recovery Expressions	29
4	Results	31
4.1	KAN training examples	31
4.1.1	Symbolification for an Example KAN	31
4.1.2	KAN behaviour near singularities	34
4.2	KANs for First Term Displacement Field Mode III	37
4.3	Hyperparameter Optimization	39
4.3.1	Hyperparameter Optimization for First Term Mode III Displacement Field	39
4.3.2	Hyperparameter Optimization for Higher Terms of Mode III Displacement Field	40
4.3.3	KAN Architecture Optimizing	42
4.3.4	Effect of Hyperparameters on Edge Usage in KANs	46

4.4	Higher Term Exploration	46
4.5	Pareto Front for different Complexity Libraries	49
4.5.1	Library with Weighted Complexity	50
4.5.2	Library with Constant Complexity	50
4.6	Synthetic Williams Field Data	52
5	Discussion	55
5.1	Symbolification Method for Edges	55
5.2	Behaviour near singularities	55
5.3	Hyperparameter Optimization	56
5.3.1	Higher Term Exploration	57
5.4	Pareto Analysis	57
5.5	Synthetic Williams Field Data	58
6	Conclusion and Outlook	59
6.1	Conclusion	59
6.2	Outlook	60
 II Appendix		
A	Libraries Pareto Front	62
A.1	library with predefined complexity	62
A.2	library with constant complexity	62
B	Recovered Formula Optuna Study	64
B.1	Recovered Formula Optuna Study	64
Bibliography		65

LIST OF FIGURES

Figure 2.1	crack opening modes, taken from Kuna, 2013, p.22 . . .	4
Figure 2.2	a) general architecture of a multi-layer perceptron, taken from James, Witten, Hastie, Trevor and Tibshirani, 2013, p. 54	9
Figure 2.3	Influence of learning rate: (a) too small, convergence takes a long time; (b) too large, convergence not achieved; (c) good value for learning rate, taken from Burkov, 2020, p. 170	11
Figure 2.4	a) unary-binary tree that represents $f(\mathbf{x}) = x_1x_2 - 2x_3$ b) sequence representation of the tree structure of $f(\mathbf{x})$, taken from Makke and Chawla, 2024, p. 4 . . .	13
Figure 2.5	[2,1,1]-KAN architecture with two layers. Highlighted is one combination of pre-activations (orange), activation functions (green), and post-activation (red).	16
Figure 2.6	[2,[1,1],1]-MultKAN architecture with two layers. Highlighted are subnodes (green), addition nodes (blue), and multiplication nodes (violet).	18
Figure 2.7	Parameterisation of an activation function as a B-spline. It is possible to switch between coarser and finer grids.	19
Figure 3.1	(a) KAN model with architecture [2, 5, 1] after training; (b) KAN model after pruning; (c) KAN model after symbolic fixation	27
Figure 4.1	The initialized KAN with three layers: the input layers with two input nodes representing x , and y , five hidden addition nodes, and one output node.	31
Figure 4.2	Training of a KAN with grid sizes = [3, 10, 20]. In (a) the symbolification was done with the <code>auto_symbolic</code> - function of the <code>pykan</code> package, whereas (b) symbolically fixes the edges manually using <code>fix_symbolic</code> (see Sec. 3.2.3. The resulting RMSE-loss is noticeably different, with (b) resulting in a lower training error. . .	32
Figure 4.3	Metrics showing parameters R^2 and complexity for the evaluating function $\mathcal{L}_{sel}$ as well as the resulting loss values.	33
Figure 4.4	Data distribution for both examined KAN models. Training data is plotted together with test data. The same data was used for both experiments.	33

Figure 4.5	Plots of the absolute error $ y_i - \hat{f}(r_i, \phi_i) $ of predictions $\hat{f}(r_i, \phi_i)$ with regards to the labels y_i . Plot (a) corresponds to the model where auto_symbolic was used, plot (b) shows the error distribution for the model with manually fixed edges.	34
Figure 4.6	scatter plot of logarithmic error (left) and absolute error (right) for $(x, y) \in [0.1, 1]$	35
Figure 4.7	scatter plot of logarithmic error (left) and absolute error (right) for $(x, y) \in [10^{-4}, 1]$	36
Figure 4.8	These plots illustrate training and test losses. For $(x, y) \in [0.1, 1]$ (a) the losses remain close, while for $(x, y) \in [10^{-4}, 1]$ training and test losses converge . . .	36
Figure 4.9	Comparison between the activation functions in the model plots for a singularity. (a) The model for $(x, y) \in [0.1, 1]$ mainly utilizes one edge, whereas the model plot (b) for $(x, y) \in [10^{-4}, 1]$ shows that both x and y contribute significantly to the output.	36
Figure 4.10	model plot after symbolification of trained edges, displacement field mode III	37
Figure 4.11	train-test error of successful symbolic regression for the first term of displacement field for mode III	38
Figure 4.12	absolute error of successful symbolic regression first term mode III	38
Figure 4.13	Optuna study on hyperparameters for the first Williams term of mode III displacement field.	41
Figure 4.14	Optuna study of hyperparameters for first and second term.	43
Figure 4.15	best trial model after symbolification in optuna study for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi)$. The plot shows several symbolically fixed linear functions, a trigonometric function was symbolically recovered for ϕ . The input from r is symbolically fixed as $\sqrt{\cdot}$	44
Figure 4.16	Optuna study of KAN architectures, specifically width and depth.	45
Figure 4.17	Models with depth 1 show significantly lower RMSEs compared to models with a higher number of layers for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$	46
Figure 4.18	model plots before pruning with different values for λ_{L1} and $\lambda_{entropy}$, with KAN architecture <code>kan_width = [2, [2, 1], 1]</code> , and data generation function $f(r, \phi) = 3\sqrt{r} \sin\frac{\phi}{2}$	47
Figure 4.19	training and test loss for higher terms $f(r, \phi) = r^{\frac{3}{2}} \sin(\frac{3}{2}\phi) + \sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi)$	48
Figure 4.20	absolute error distribution of the model predictions with regards to the labels for higher terms of mode III	48

Figure 4.21	model plot for higher terms of displacement fields for mode III after symbolic fixing	49
Figure 4.22	Pareto front for weighted complexity terms c_{kan} . “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow.	51
Figure 4.23	Pareto front for weighted complexity c_{func} for the reduced expressions. “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow. Models that resulted in the same expression structure as the underlying function for data generation are marked red.	51
Figure 4.24	Pareto front for constant complexity values c_{kan} . “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow.	52
Figure 4.25	Pareto front for constant complexity values c_{func} for the reduced expressions. “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow. Models that resulted in the same expression structure as the underlying function for data generation are marked red.	53
Figure 4.26	Distribution of simulated data for the displacement field u_x . (a) shows the 2D distribution in the x - y plane, while (b) shows the corresponding 3D representation	53
Figure 4.27	(a) shows the prediction of the displacement field u_x by a KAN, while (b) shows the absolute error of the prediction for displacement field u_x w.r.t labels	54
Figure 4.28	model architecture for KAN model trained on simulated data for displacement field u_x	54

LIST OF TABLES

Table 3.1	given regularization parameters for Study C	28
Table 4.1	hyperparameters for successful SR of first williams term of displacement field in mode III	39
Table 4.2	optimized hyperparameters for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$.	40
Table 4.3	optimized hyperparameters for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right) +$ $r \cos(\phi)$	42
Table 4.4	optimized parameters for KAN architecture and prun- ing threshold	44
Table 4.5	The 7 most successful models for the weighted library, according to RMSE. c_{kan} is the number of active edges after symbolic fixing the model, c_{func} the complexity of the expression with respect to the original pykan library.	50
Table 4.6	The 7 most successful models for the constant library, according to RMSE. c_{kan} is the number of active edges after symbolic fixing the model, c_{func} the complexity of the expression with regards to the original pykan library. Because the length of some expressions ex- ceeds the available space they are only hinted at.	52

ACRONYMS

AI	Artificial Intelligence
DLR	German Aerospace Center
GP	Genetic Programming
KAN	Kolmogorov-Arnold network
LEFM	Linear Elastic Fracture Mechanics
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Square Error
NaN	numerical error
NN	Neural Network
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
RSS	Residual Sum of Squares
SR	Symbolic Regression
TSS	Total Sum of Squares

Part I

THESIS

INTRODUCTION

1.1 BACKGROUND AND CONTEXT

In natural sciences Artificial Intelligence (AI) tools are developed at an increasing pace. Additionally, AI promises to influence many aspects in society, both in professional settings and at home [8]. As a subfield of AI, Machine Learning (ML) evolved to extract patterns from data and derive knowledge on their own[7]. This capability is used to make predictions and make decisions based on large amounts of data [21]. However, many machine learning models do not explain their decision to users in an understandable way. They are commonly referred to as “black-box” models. There are high-stakes environments where interpretability is necessary, and “black-box” models are detrimental [22].

Symbolic Regression (SR), which can be seen the discovery process of desired outputs that are inferred from particular inputs [11], can be used to “white-box” a “black-box” model [16].

In the 17th century, Kepler developed his model for the motion of planets using years of astronomical observations [26]. His data-driven trial and error approach, where he explicitly showed calculations that did not work[10], is still used in symbolic regression as a model discovery tool [16]. The main goal of SR is to automatically derive or recover natural relations from data, without imposing restrictions on their form [23].

In 2024 Liu et al.[15] published a working paper which has since been cited over 4000 times. In their paper they propose a new type of Neural Network (NN), the so-called Kolmogorov-Arnold network (KAN), that, among other things can be used for symbolic regression. A comparison paper between KANs and Multilayer Perceptron (MLP) found that KANs outperform MLPs for symbolic formula representation tasks [29].

In 1957, Williams was able to derive an eigenfunction expansion for isotropic linear-elastic materials, the so-called Williams series, which enabled the description of the asymptotic crack-field (stress and displacement fields) near the crack tip [28] [17]. At the German Aerospace Center (DLR), explainable AI is used, among other things, for detection of crack tips in materials [19].

1.2 RESEARCH QUESTIONS

The motivation for this master thesis was to investigate whether Kolmogorov-Arnold networks can successfully recover parts of the infinite Williams series,

and look into whether they can be used for scientific discoveries in material sciences, with a focus on fracture mechanics. For this purpose the following research questions is explored: Can the basic building blocks of the Williams series be reliably recovered from data with the use of KANs?

Existing approaches use multiple models to reduce the complexity of the output function. That way they reduce the Architecture of neural networks with every iteration [4], or iteratively build models in genetic programming [16]. This thesis wants to investigate, whether the KAN architecture can be used successfully training only one model.

1.3 STRUCTURE OF THE THESIS

The following master thesis is structured in six chapters. Chapter 2 introduces the theoretical background, including an introduction into symbolic regression and Kolmogorov-Arnold networks. Chapter 3 explains the methods used in this thesis, including data generation and it describes the workflow of KANs.

In chapter 4, the results of the work are presented. This includes exploration of examples presented in [15], optimizing hyperparameter, and testing different libraries for symbolic regression.

The results are analyzed in chapter 5, followed by the conclusion of the thesis and an outlook into possible future research in chapter 6.

THEORETICAL BACKGROUND

2.1 LINEAR ELASTIC FRACTURE MECHANICS

If not quoted otherwise, the information of this section is taken from Kuna (2013) [13].

Fracture mechanics is the field that researches fracture and failure processes both in materials. A core assumption of fracture mechanics is the presence of flaws or other defects in every material and every component. Such defects are modeled as cracks.

A crack can have different modes of deformation, so-called *crack opening modes*. They are shown in Fig. 2.1.

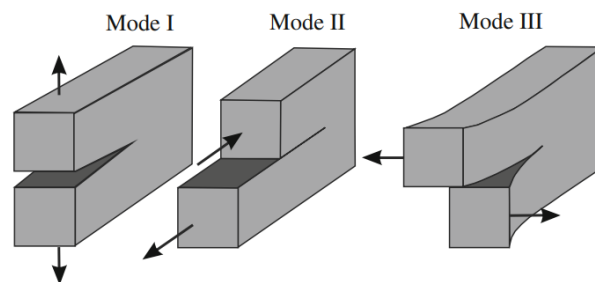


Figure 2.1: crack opening modes, taken from Kuna, 2013, p.22

The three modes have the following definitions:

- Mode I: Opening Mode - The opening of the crack happens vertically.
- Mode II: In-plane sliding mode - Displacement happens in the plane of the crack faces. usually towards the crack front.
- Mode III: Out-of-plane tearing mode - Parallel to the crack front, a displacement of the crack faces happens.

These three basic kinematic modes happen with every crack deformation. Therefore each deformation can be interpreted as a superposition of the modes.

In Linear Elastic Fracture Mechanics (LEFM), deformation behavior that can be assumed to be linear-elastic. For elastic materials, this means they have two features:

- The deformations are reversible.

- Neither time nor loading rate have an influence on the deformations.

Hooke's Law states that elastic deformation behaviour is linear

$$\sigma = E\epsilon, \quad (2.1)$$

where σ is the stress, ϵ represents the elastic strain, and E the modulus of elasticity. In LEFM, it is assumed that the deformation behavior of materials, for which the crack problems are explored, can be treated as linear-elastic, according to Eq. 2.1.

2.2 CRACK TIP FIELD AND WILLIAMS SERIES

If not quoted otherwise, the information of this section is taken from Kuna (2013) [13].

For all three modes there are different solutions to describe the crack tip field, also called an asymptotic near field. The solutions are dependent on the boundary conditions.

2.2.1 Williams Series

The linear-elastic stress and displacement fields around a crack tip can be described by the Williams series expansion[28]. It applies to all plane elastic crack problems. Additionally, the complete infinite expansion is defined for a two-dimensional homogeneous and isotropic body[25]. The series can be written as:

$$\sigma_{ij}(r, \theta) = \sum_{n=1}^{\infty} r^{\frac{n}{2}-1} \left[a_n \tilde{M}_{ij}^{(n)}(\theta) + b_n \tilde{N}_{ij}^{(n)}(\theta) + c_n \tilde{L}_{ij}^{(n)}(\theta) \right], \quad (2.2)$$

$$u_i(r, \theta) = \frac{1}{2\mu} \sum_{n=1}^{\infty} \left[a_n \tilde{F}_i^{(n)}(\theta) + b_n \tilde{G}_i^{(n)}(\theta) + c_n \tilde{H}_i^{(n)}(\theta) \right]. \quad (2.3)$$

Here $\sigma_{ij}(r, \theta)$ represents the stress field around the crack tip, whereas $u_i(r, \theta)$ denotes the displacement field. The coefficients a_n , b_n and c_n are unknown, and depend on the boundary conditions of the respective crack problems. $\tilde{M}_{ij}^{(n)}(\theta)$, $\tilde{N}_{ij}^{(n)}(\theta)$ and $\tilde{L}_{ij}^{(n)}(\theta)$ represent trigonometric eigenfunctions, that represent the stress tensors[18]. $\tilde{F}_{ij}^{(n)}(\theta)$, $\tilde{G}_{ij}^{(n)}(\theta)$ and $\tilde{H}_{ij}^{(n)}(\theta)$ are the eigenfunctions for the displacement field.

Of particular interest for this thesis, as it is the function used as a basis for most calculations and training, is the displacement field for mode III:

$$u_3^{(n)}(r, \theta) = \frac{c_n}{2\mu} r^{\frac{n}{2}} \tilde{H}_3^{(n)}(\theta), \quad \tilde{H}_3^{(n)}(\theta) = \begin{cases} 2 \sin \frac{n}{2} \theta & \text{for } n = 1, 3, \dots \\ 2 \cos \frac{n}{2} \theta & \text{for } n = 2, 4, \dots \end{cases} \quad (2.4)$$

2.3 MACHINE LEARNING

As a subfield of computer science, machine learning (ML) concerns itself with building algorithms. Part of that can be to use knowledge gained from datasets to make predictions. It can also be described as a process, in which a problem is solved by first sampling data, and afterwards training statistical models on the sampled data [21, p.1][9, p.1][3, p.4].

Generally, machine learning is categorized in three different types: supervised learning, unsupervised learning, and reinforcement learning [21, p.2], although there are models that utilise both supervised and unsupervised learning. They are referred to as semi-supervised [3, p.4]. The thesis only utilizes supervised learning, which is why it is explained in detail.

2.3.1 Supervised Learning

In supervised learning, the data is distinguished between input data, also called features, and output data, referred to as label, or sometimes dependent variable. Since the output is already part of the dataset, it is called labeled data. The methods are called supervised, because in training the desired outputs are already known [21, p.4][9, p.26][3, p.4].

If the goal of a model is to predict in which discrete class to categorize a new input, the method is called a classification. Classic examples for classification are diagnosing illnesses, or detect spam emails [21, p.4][3, p.4-5].

On the other hand, a model can predict continuous outcomes. Here the problem is to predict a number. This type of supervised learning is called regression. There are predictor variables, the input data, that is used to predict the response(output) of future observations. An example for regression analysis is the prediction of an employees salary based on work experience and their knowledge [21, p.5][9, p.26][3, p.4].

2.3.1.1 Linear Regression

As already established, regression analysis examines problems with quantitative responses, which means the output is a numerical value [9, p. 28]. The parameters are fitted to an equation of a set form [23]. A linear relationship between Y and X can be written as

$$Y \approx \beta_0 + \beta_1 X, \quad (2.5)$$

where X is a single predictor variable, and Y is the quantitative response, that the model tries to predict. β_0 and β_1 are unknown constants. They show the intercept, which is the interception point with the y-Axis, and the slope of the graph respectively. Based on training data the model coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ are estimated, and a prediction model can be formulated:

$$\hat{y} = \hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x. \quad (2.6)$$

Variables with the hat symbol denote either estimated values of unknown parameters/coefficients or they represent the predicted output value. Here, $\hat{f}(x)$ stands for the prediction of Y . It is based on $X = x$ [9, p. 61].

To evaluate the performance of a linear regression model $\hat{f}(x)$ a metric is needed. The most commonly used metric is the Mean Square Error (MSE), which is given by

$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^n \left(y_i - \hat{f}(x_i) \right)^2, \quad (2.7)$$

where y_i is the i th label of the data, and $\hat{f}(x_i)$ represents the prediction for the i th observation. For a prediction, that is close to the true label the MSE will return a small value. On the other hand, if the prediction differs drastically from y_i , the MSE will be large [9, pp. 29–30].

The goal for a model is that it is applicable on unseen data. The data set used to develop the model is called training data, and data used to test the performance of the model is called test data. For both we can calculate their errors respectively. Two challenges arise when trying to develop algorithms: on one hand we want small discrepancies between test and training errors, on the other hand we generally want to reach a sufficiently small error in our training data set. Underfitting occurs when the model does not manage to reach a small enough error value for on the training data. Overfitting on the other hand means when the model yields a small MSE in training, but for the test data the MSE is too large [9, p. 32][7, p. 108].

An example for a measure of fit is the R^2 score, which explains the proportion of variance in Y :

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}. \quad (2.8)$$

$TSS = \sum (y_i - \bar{y})^2$ represents the total sum of squares, whereas $RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ is the residual sum of squares. R^2 is a proportion and takes on values between 0 and 1. It indicates which ratio of variability in Y can be explained by using X . A value close to 1 indicates the regression can explain a large proportion in the responses variability, whereas a value close to 0 shows

that the variability was not explained much by the regression. As a general rule, the higher R^2 the better is the model, although it is dependent on the circumstances what constitutes a good R^2 score [9, pp. 69–70]

2.3.2 Multi-layer Perceptrons

Neural networks are graphs, that consist of vertices (neurons), and edges (connections). Each edge has a weight associated with it. Each neuron is assigned three quantities: input, activation, and output. Furthermore, neurons belong to one of three categories: input neurons, output neurons, and hidden neurons. Input neurons receive an input from the environment, whereas output neurons release an output into the environment. Hidden neurons receive and give output to different neurons, and have therefore no contact with the environment [12, p. 40].

For a MLP, the weighted sum of all inputs is the input function of each hidden and output neuron. The weighted sum incorporates the weights on the edges. For node j in the first hidden layer, the output can be formulated as

$$y_j(p) = \sigma \left(\sum_{i=1}^n x_i \cdot w_{ij} - \theta_j \right), \quad (2.9)$$

where n is the total number of inputs that reach the node. σ denotes the activation function, w_{ij} is the weight of the connection from input node i to output node j . x_i is input i and θ_j is a bias value, that acts as a threshold for the activation function [20]. The bias influence can be seen for example in a step function [9, p. 44]

$$f(x, \theta) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{otherwise} \end{cases} \quad (2.10)$$

Connections of a MLP only exist between consecutive layers, which means each node of the input layer connects with each node of the first hidden layer, each node of the first hidden layer connects with each node of the following layer, and so forth [12, pp. 53–54]. Consequently, MLPs are also referred to as feedforward Neural Networks. In Fig. 2.2 the architecture for a L -layered perceptron is depicted. It has an input layer l_{in} , an output layer l_{out} , and $L - 2$ hidden layers. If a network has more than one hidden layer, it is called a deep neural network [21, pp. 338–339].

A complete MLP with one hidden layer can be formulated as

$$\hat{\mathbf{y}} = \sigma \left(\mathbf{w}^{(2)\top} \mathbf{h}^1 + \theta^2 \right) \quad (2.11)$$

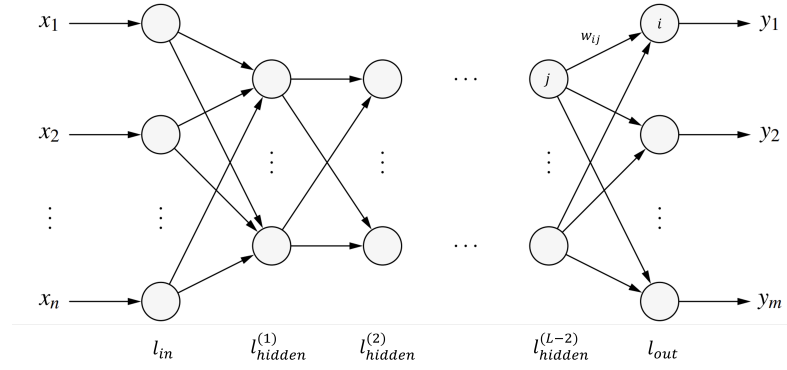


Figure 2.2: a) general architecture of a multi-layer perceptron, taken from James, Witten, Hastie, Trevor and Tibshirani, 2013, p. 54

$$\mathbf{h}^1 = \sigma \left(\mathbf{w}^{(1)\top} \mathbf{x} + \theta^1 \right), \quad (2.12)$$

with $\hat{\mathbf{y}}$ representing the output layer, and $\mathbf{h}^1$ the (first, and only) hidden layer. $\mathbf{w}^{(1)\top}$ are the vectorised weights, and $\mathbf{x}$ is the input vector [7, pp. 191–192]. Variables that are displayed in bold are vectors or matrices.

2.3.2.1 Universal Approximation Theorem

The process of finding underlying relationships between inputs and output of a data set is called function approximation [30]. As the most common neural network, MLPs use the Universal Approximation Theorem to approximate functions. It states that for continuous, bounded, and non-constant functions $\varphi(\cdot)$ the difference between the output of the MLP and the function the network tries to approximate is smaller than an error bound ϵ . The approximation can be defined as

$$F(\mathbf{x}) = \sum_{i=1}^N v_i \sigma(\mathbf{w}_i^\top \mathbf{x} - \theta_i), \quad (2.13)$$

so that

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon,$$

for all $\mathbf{x} \in X$ and $\epsilon > 0$. X denotes a compact subset of $\mathbb{R}^m$, and $C(X)$ the space of continuous functions on X . σ represents the activation functions of the hidden neurons, the weights of the input neurons to the neurons in the hidden layer are given by $\mathbf{w}_i \in \mathbb{R}^m$. Furthermore, the constants v_i and θ_i denote the weights of the hidden layer to the output layer and the biases of the hidden neurons respectively. This shows that for non-linear activation functions one hidden layer suffices to approximate functions sufficiently well. However, for an MLP with more hidden layers it may be simpler to compute

functions of interest, so there are arguments for initializing neural networks of more than one hidden layer [12, pp. 66–67].

2.3.2.2 Gradient Descent and Error Backpropagation

Gradient descent is a method used in MLPs to update weights and biases to reduce the cost function. For that to be possible, we choose differentiable activation functions. They need to be differentiable, because the gradient depicts the rate of change for functions for each input. It is a vector composed of partial derivatives for each input. The gradient ∇f for a function $f(x_1, x_2)$ can be written as

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right]. \quad (2.14)$$

Notably, commonly used Rectified Linear Unit (ReLU)s have an activation function

$$g(x) = \max\{0, x\}, \quad (2.15)$$

which is non-differentiable in $x = 0$. In practice, the value for $g'(x)$ can be chosen arbitrarily between $[0, 1]$. PyTorch and TensorFlow, popular libraries for machine learning applications, both use $g'(x) = 0$ [2].

The gradient descent method starts at a random point in the search space. Afterwards steps are made along the steepest descent. This process is repeated iteratively until a pre-defined termination criterion is fulfilled. This can be the number of steps that have been executed, or the value of the current gradient is small [12, pp. 72–73]. The size of the steps taken is controlled by the learning rate ν . Picking the correct value for ν is important, as for learning rates that are too big, the values might not converge. However, for too small values the time it takes to reach the minimum takes is very long or, depending on the termination criterion, the minimum might not even be reached. The differences are shown in Fig. 2.3.

Error Backpropagation is used to update the weights in a MLP. Based on the error $\mathcal{L}$ (see Eq. 2.3.1.1) that is calculated after the neural network computes an output $\hat{y}$ (see Eq. 2.3.2, the gradient of the output layer is calculated:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i} (y_i - \hat{y}_i)^2 = 2(\hat{y}_i - y_i). \quad (2.16)$$

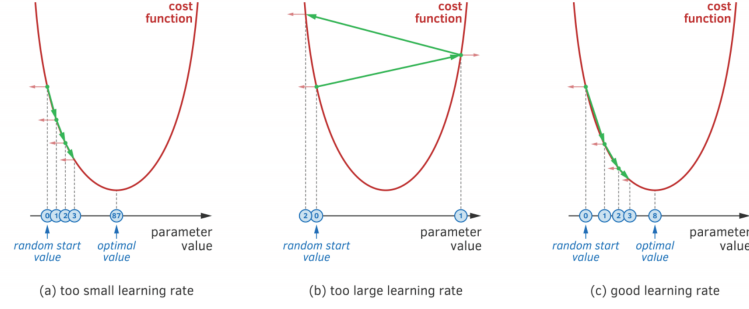


Figure 2.3: Influence of learning rate: (a) too small, convergence takes a long time; (b) too large, convergence not achieved; (c) good value for learning rate, taken from Burkov, 2020, p. 170

$\hat{y}_i$ is the i th output node, and dependent on the activation function σ . Similarly to Eq. 2.3.2.2 calculate the partial derivative for the activation function σ is represented as

$$\frac{\partial \hat{y}_i}{\partial \sigma} = \frac{\partial}{\partial \sigma} \hat{y}_i(\sigma). \quad (2.17)$$

The result depends on the activation function σ . To adjust the weight w_{ij} of node j in the hidden layer to node i in the output layer, another partial derivation is calculated (see Fig. 2.2:

$$\frac{\partial \sigma}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} w_{ij} \mathbf{h} + \theta_i. \quad (2.18)$$

For updating weight w_{ij} , three chained partial derivations are necessary.

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial w_{ij}} \quad (2.19)$$

The calculated value of the gradient for the weight is used to upgrade the weight with the learning rate v :

$$w_{ij} := w_{ij} - v \frac{\partial \mathcal{L}}{\partial w_{ij}} \quad (2.20)$$

Error backpropagation is used to update every weight and bias. Afterwards the network is given the input variables again and the outputs are calculated with new weights and biases. The loss function is calculated and the process is repeated until a termination criterion is reached [21, pp. 362–366][7, pp. 205, 206].

2.3.3 Regularization

So far, the objective function that models try to optimize has been the loss function. To combat overfitting and models that are too complex, additional modification can be added to the training error, so that a new optimization metric can be formulated [7, p. 117].

To adjust the objective function Eq. 2.3.1.1 is expanded:

$$\mathcal{L}_{obj} = \frac{1}{n} \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j|. \quad (2.21)$$

This objective function uses $L1$ regularization, which estimates some coefficients β_j to 0 for sufficiently high λ . This leads to a sparsification of the model, as the influence of some features x_j is then also reduced to 0 [9, p. 219]. For $\lambda = 0$ the objective function reduces to the MSE. λ is called a *hyperparameter*. Hyperparameters are settings the user chooses that are used to control the behaviour of an algorithm [7, p. 117].

2.4 SYMBOLIC REGRESSION

Contrary to linear regression, where parameters are fitted to a specified linear model (and therefore a linear formula), symbolic regression (SR) looks for both parameters and the formula simultaneously. The goal for SR is to make models interpretable, which means the relationship between predictors and responses can be understood logically or mathematically. Models that do not present users with tools to understand that relationship are called “*blackbox*”. An example for “*blackbox*” models are NNs, that have several layers of non-linear transformations, which hinders interpretability. SR models on the other hand link outputs directly to inputs [16] by providing a mathematical expression, that consists of different units like operators (+, −, ·, ÷), functions (e.g. sine, log), or constants. In an algorithm, new equations are formed by adjusting previous ones and disregarding unpromising solutions. After reaching a desired level of accuracy, the algorithm returns an equation that is most probable to model the underlying relationships in the observed data set [23].

Makke and Chawla [16] define SR as similar to classical Empirical Risk Minimization:

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $\mathbf{x}_i \in \mathbb{R}^d$ being the input vector and $y_i \in \mathbb{R}$ being a scalar output. Let $\mathcal{F}$ be a function class consisting of mappings $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Additionally, define a loss function

$$l(f) := \sum_{i=1}^n l(f(\mathbf{x}_i), y_i)$$

for every candidate $f \in \mathcal{F}$. One of the most common loss functions is the MSE. The minimization task is to look for the function f out of the discrete set F , that minimizes the loss function

$$f^* = \arg \min_{f \in \mathcal{F}} l(f).$$

Often times a complexity measure for the equation is combined with the loss function, which penalizes longer or more complex solutions. As a consequence sparser expressions get preferred by the algorithm [4].

Symbolic expressions can be represented in several different forms. A common way to represent the solution is in form of a tree-like structure, as shown in Fig. 2.4a. Fig. 2.4b shows a sequence representation of the same equation $f(\mathbf{x}) = x_1x_2 - 2x_3$, where the tree is traversed "top to bottom and left to right in a depth-first matter" [16, p. 4]. The symbolic expression represented in KANs is similar to the tree-like structure. A library L is necessary to compose the symbolic regression. In such a library many basic mathematical functions may be included, like square root, logarithm, exponential function, and trigonometric functions. In theory there are infinitely many functions that could be included in a library, which makes some form of pre-selection necessary. If prior domain knowledge is available, the library can be reduced to the relevant operations, which reduces the search space [16].

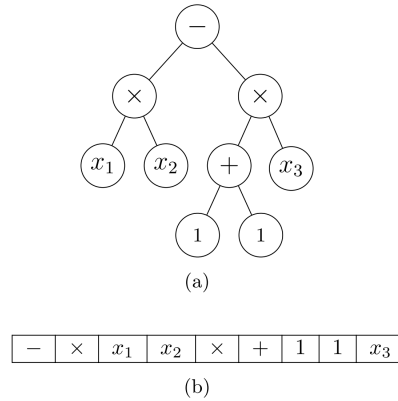


Figure 2.4: a) unary-binary tree that represents $f(\mathbf{x}) = x_1x_2 - 2x_3$ b) sequence representation of the tree structure of $f(\mathbf{x})$, taken from Makke and Chawla, 2024, p. 4

2.4.1 Symbolic Regression Methods

Popular methods for symbolic regression are regression-based methods, expression tree-based methods, and physics-inspired methods.

Regression-based methods can be divided in linear and non-linear models. Linear models can be denoted as

$$f(x, \theta) = \sum_j \theta_j h_j(x),$$

where θ_j represents a coefficient that puts a weight to an unary operator $h_j(\cdot)$ taken from the library L . j spans the functions of L , and x denotes the input vector. In linear symbolic regression methods a single model structure is imposed. That makes it deterministic, but also rigid in its expression. Non-linear models have their structure defined by a NN. They can be written as:

$$f(x, W, b) = \sigma \left(\sum_i W_i \sigma \left(\sum_j W_j \sigma \left(\cdots \sum_\ell W_\ell \sigma \right) \right) \right) + b,$$

with σ being a nonlinear activation function, W_{idx} are learnable parameters in layer idx . The optimization task is to find parameters $\{W, b\}$ for the neural network layers.

Expression tree-based methods include Genetic Programming (GP). GP is based on Darwinian natural selection. Several hundreds of computer programs are measured on their *fitness*. Often time the fitness is calculated by the error of the program. The closer to zero the error, the fitter the program. Well performing programs are used as parental computer programs to reproduce into new *offspring programs*. With randomly chosen parts of the well performing parental programs being recombined into new programs, there may be even fitter programs being generated [11]. Thus the loss function becomes minimized. An example for a GP-based method is the commercially available Eureqa [5], that is often used as a baseline method for research [16].

As a physics-inspired SR method, *AlFeynman* [27] uses a set of solvers (dimensional analysis, polynomial fit, and brute force search) to find solutions for SR problems. The different solvers are applied recursively. Should the problem remain unsolved, the data is analyzed for properties, that simplify the task. These properties are used recursively to reduce the dataset in simpler sub-problems, that have fewer independent variables. Then SR methods are applied on the sub-problems [16].

Since KANs use a similar approach as symbolic regression methods for non-linear models with neural networks [15], this thesis focuses on them.

2.5 KOLMOGOROV-ARNOLD NETWORK

The Kolmogorov-Arnold representation theorem states that, if f is a continuous multivariate function $f : [0, 1]^n \rightarrow \mathbb{R}$, f can be written as follows:

$$f(\mathbf{x} = f(x_1, \dots, x_n)) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right), \quad (2.22)$$

which is a finite composition of continuous univariate functions the binary operation of addition. So multidimensional functions can be represented as a set of 1D-functions [15].

Kolmogorov-Arnold Networks(KANs) were first proposed by Liu et al.[15] as a promising alternative to multi-layer perceptrons (MLPs). One key difference is the location of the activation functions: while MLPs have their fixed activation functions on neurons, KANs on the other hand place their respective activation function on edges. Those activation functions are learnable, similar to weights on the edges of MLPs[15]. Each edge in a KAN represents a learnable 1D-function (see Fig. 2.5. They are parameterized as a spline.

2.5.1 KAN Architecture

The information in this section is taken from this source: [15] In their current form, KANs provided by the authors who wrote of both the original paper and the python package *pykan* have a dual structure per layer: the first sublayer is a *KAN layer*, the second sublayer is a *Mult layer*. Together they form a *MultKAN layer*. To properly introduce the architecture of a so-called MultKAN, this thesis will start with an explanation of the original KAN implementation, and extend it to the current version thereafter.

2.5.1.1 Original KAN Architecture

A KAN consists of layers. Each layer has n_{in} input nodes, and n_{out} output nodes. For each combination of input and output node exists an activation function ϕ . Thus, the whole layer can be defined as a matrix

$$\Phi = \{\phi_{j,i}\}, \quad j = 1, 2, \dots, n_{out}, \quad i = 1, 2, \dots, n_{in}. \quad (2.23)$$

The shape of an original KAN architecture the is expressed as followed:

$$[n_0, n_1, \dots, n_L]. \quad (2.24)$$

The number of neurons in the l^{th} layer is represented by n_l , with L being the entirety of layers. In layer l are n_l nodes, of which the i^{th} neuron is denoted as (l, i) . The neuron has an activation value $x_{l,i}$. Since activation functions connect each node of the layer l with each node of layer $l + 1$, there are $n_l n_{l+1}$ total activation functions in a KAN layer. The notation for the individual functions is denoted as

$$\phi_{l,j,i}, \quad l = 0, \dots, L-1 \quad j = 0, \dots, n_{l+1}-1 \quad i = 0, \dots, n_l-1. \quad (2.25)$$

The initial index for the indices is 0, which leads to a shift by 1. See Fig. 2.5 for a visual explanation. The activation functions are marked in green.

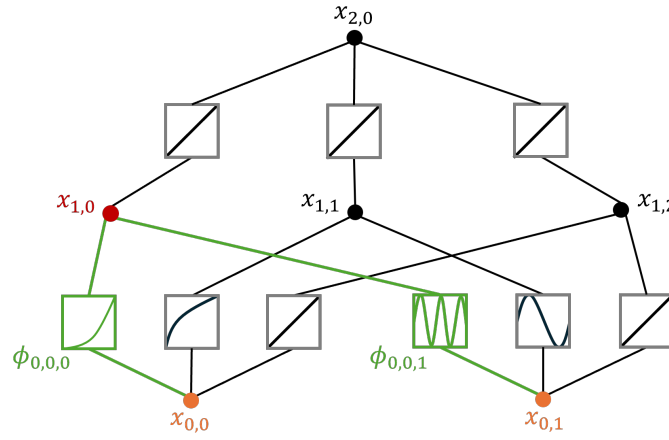


Figure 2.5: [2,1,1]-KAN architecture with two layers. Highlighted is one combination of pre-activations (orange), activation functions (green), and post-activation (red).

The input for $\phi_{l,j,i}$ is $x_{l,i}$, the so called pre-activation (highlighted in orange in Fig. 2.5). The output of $\phi_{l,j,i}$ is also called post-activation and is written as $\tilde{x}_{l,j,i} \equiv \phi_{l,j,i}(x_{l,i})$. Since the j^{th} neuron in layer $l + 1$ receives output of all incoming activation functions, its own activation value can be calculated as the sum of the post-activation (highlighted red in Fig. 2.5):

$$x_{l+1,j} = \sum_i \tilde{x}_{l,j,i} = \sum_i \phi_{l,j,i}(x_{l,i}), \quad j = 1, \dots, n_{l+1}. \quad (2.26)$$

This can also be written in matrix form:

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\Phi_l} \mathbf{x}_l \quad (2.27)$$

Φ_l represents the function matrix describing the l^{th} KAN layer. In general, a KAN network is a structure made out of L layers. The output is

$$\text{KAN}(\mathbf{x}) = (\Phi_L - \mathbf{1} \circ \Phi_{L-2} \circ \dots \circ \Phi_1 \circ \Phi_0) \mathbf{x}, \quad (2.28)$$

with $\mathbf{x} \in \mathbb{R}^{n_0}$ being the input.

2.5.1.2 MultKAN Architecture

In an additional, later published paper the KAN architecture proposed in [15] was updated to a MultKAN architecture[14]. This section is based on both the original and the follow-up paper.

The difference between the architectures is that the new network incorporates both addition as well as multiplication operations, whereas the original was limited to the former. Still, MultKANs contain standard KAN layers Φ_1 described in the previous section. However, each layer l in a MultKAN also contains a multiplication layers $\mathbf{M}_l$, so that a MultKAN layer is comprised of both a KAN layer and a multiplication layer. The complete layer can be written as $\Psi_l \equiv \mathbf{M}_l \circ \Phi_l$ (see Fig. 2.6).

In this architecture, the input nodes or pre-activation nodes of a KAN layer are referred to as “nodes”, whereas output nodes are now called “subnodes”.

A multiplication layer has two different transformations. If the nodes are supposed to be addition nodes, they are copied from the previous layer’s subnodes. Multiplication nodes on the other hand perform a multiplication, where k subnodes are multiplied with each other. For explainability purposes, k is set to 2 (see Fig. 2.6).

A KAN layer consists of nodes $x_{l,i}$, activation functions $\phi_{l,j,i}$ and outputs $z_{l,j}$. $x_{l,i}$ is the i^{th} neuron in the l^{th} layer, $z_{l,j}$ is the j^{th} subnode in the l^{th} layer. $z_{l,j}$ can be calculated by:

$$z_{l,j} = \sum_{i=0}^{n_l} \phi_{l,j,i} x_{l,i} \quad (2.29)$$

All nodes in a KAN layer Φ_l can be represented by an input vector $\mathbf{x}_l \in \mathbb{R}^{n_l^a + n_l^m}$. The subnodes are $\mathbf{z}_l \in \mathbb{R}^{n_{l+1}^a + 2n_{l+1}^m}$. The factor 2 in front of n_{l+1}^m is a result of the multiplication operation: to get one input for the $l+1^{th}$ layer two subnodes get multiplied (see violet node in Fig. 2.6).

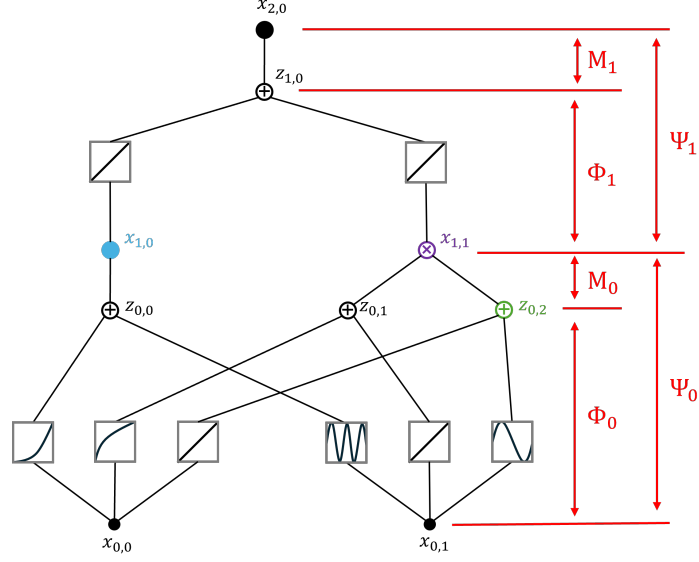


Figure 2.6: $[2,[1,1],1]$ -MultKAN architecture with two layers. Highlighted are subnodes (green), addition nodes (blue), and multiplication nodes (violet).

Addition and multiplication operations in layer l are denoted n_l^a and n_l^m , respectively. For the whole network, the notation is the following: addition width $\mathbf{n}^a \equiv [n_0^a, n_1^a, \dots, n_L^a]$, multiplication width $\mathbf{n}^m \equiv [n_0^m, n_1^m, \dots, n_L^m]$. Should $n_0^m = n_1^m = \dots = n_L^m = 0$, the MultKAN reduces to a network consisting only of KAN layers, which is the same as a traditional KAN. The shape of a MultKAN with depth L is represented by an array

$$[[n_0^a, n_0^m], [n_1^a, n_1^m], \dots, [n_L^a, n_L^m]], \quad (2.30)$$

where each entry consists of a tuple with the amount of both addition and multiplication operations in the concerning layer.

2.5.1.3 Activation function

An activation function is the sum of a base function $b(x)$ and a spline function $B_{tot}(x)$. It is defined as

$$\phi(x) = w_b b(x) + w_s B_{tot}(x). \quad (2.31)$$

The spline function B_{tot} is a linear combination of B-splines:

$$B_{tot} = \sum_i c_i B_i(x). \quad (2.32)$$

The parameter c_i , as well as the factors w_b and w_s are trainable. During the training of a KAN one might want to increase the numbers of c_i . This is called fine-graining spline grids, or grid extension, and results in more B-splines in an activation function. This can make KANs more accurate. In Fig. 2.7 the effects of grid extension is illustrated. Since with $k = 3$ (here k means the order of the splines, $k = 3$ equates to a cubic spline) three B-splines $B_i(x)$ are necessary to approximate the underlying formula, there are $G + k$ splines necessary for G intervals ($G + 1$ grid points).

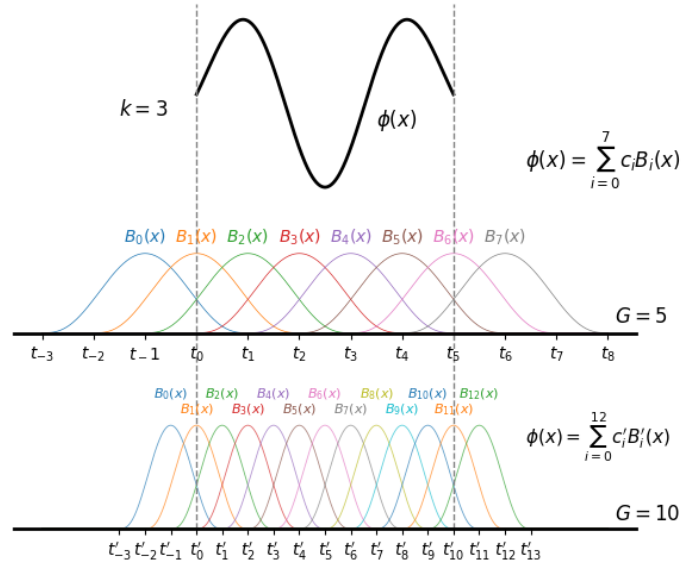


Figure 2.7: Parameterisation of an activation function as a B-spline. It is possible to switch between coarser and finer grids.

2.5.1.4 B-Splines

B-splines are shown in Fig. 2.7 for $k = 3$. A B-spline of degree k has the following properties:

- A B-spline consists of $k + 1$ polynomial pieces. Each piece is of degree k .
- The polynomial pieces of a B-spline join at k inner knots.
- B-splines are positive on a domain spanning $k + 2$ knots; they are zero everywhere else.
- For any given x -value, $k + 1$ B-splines are non-zero.

B-splines with $k = 3$ consist of four cubic pieces, that are joined at three knots. The other two knots give the boundaries of the B-Splines. At the knots,

the derivatives of each piecewise cubic function are the same. B-Splines are popular as base functions for regression tasks, because a linear combination of third-degree B-splines can produce smooth curves [6].

2.5.2 Regularization

This section is based largely on the original paper by Liu et al[15]. Other sources are mentioned accordingly.

To favor sparsity in a MLP, a common idea is to introduce L1-regularization. KANs adapt this method, although a few modifications are needed. Since there are no linear weights in KANs, but learnable activation functions, the L1 norm of these functions is defined instead. Additionally, in KANs L1 prove to be not sufficient enough for sparsification. Therefore it is necessary to introduce an additional regularization, in this case entropy regularization. This regularization technique reduces the active activation functions. Without it, there would be too many duplicate functions.

The complete regularization term is the following:

$$\mathcal{L}_{\text{reg}} = \lambda_{\text{reg}}(\lambda_{\text{L1}}\mathcal{L}_{\text{L1}} + \lambda_{\text{entropy}}\mathcal{L}_{\text{entropy}} + \lambda_{\text{coef}}\mathcal{L}_{\text{coef}} + \lambda_{\text{coefdifff}}\mathcal{L}_{\text{difference}}), \quad (2.33)$$

where λ_i are hyperparameters. λ_{reg} controls the overall regularization strength, whereas λ_{L1} , λ_{entropy} , λ_{coef} , and λ_{diff} weight the contributions of the individual regularisation terms. In Sec. 4.3.4 there will be a more detailed look into the regularization contributions of λ_{coef} and λ_{diff} .

Liu et al. proposed to “define the L1 norm of an activation function ϕ to be its average magnitude over its N_p inputs, i.e.”

$$|\phi|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} |\phi(x^{(s)})|. \quad (2.34)$$

The L1 norm for a whole KAN layer Φ is the sum of the L1 norms of all activation functions in that layer

$$\mathcal{L}_{\text{L1}} = |\Phi|_1 \equiv \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} |\phi_{i,j}|_1, \quad (2.35)$$

where n_{in} are the inputs, and n_{out} are the outputs of the corresponding activation function.

The entropy of Φ can be defined as

$$\mathcal{L}_{\text{entropy}} = S(\Phi) \equiv - \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} \frac{|\phi_{i,j}|_1}{|\Phi|_1} \log\left(\frac{|\phi_{i,j}|_1}{|\Phi|_1}\right). \quad (2.36)$$

Alternatively, a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be constructed, with A_{ji} representing the aforementioned average magnitude

$$A_{ji} = |\phi_{j,i}(x_i)|, \quad (2.37)$$

where x_i , $i \in n$ input, and $j \in m$ as output dimension. This can be interpreted as "capturing the degree to which each input dimension contributes to the output between samples"([4], S.5).

$\mathbf{A}$ can also be used to encourage sparsity when used to calculate the entropy. For that, both column-wise and row-wise entropies ($\mathcal{H}_{\text{col}}, \mathcal{H}_{\text{row}}$) are computed. They are a measure for the diffuseness of activations across samples and inputs. With

$$p_{ji}^{\text{col}} = \frac{A_{ji}}{\sum_{j=1}^m A_{ji} + \epsilon}, \quad p_{ji}^{\text{row}} = \frac{A_{ji}}{\sum_{i=1}^n A_{ji} + \epsilon}, \quad (2.38)$$

where $\epsilon > 0$ is a small value to prohibit division by 0, and staying in the defined space of logarithm, the entropy loss can be defined as follows:

$$\mathcal{H}_{\text{col}}(A) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m p_{ji}^{\text{col}} \log(p_{ji}^{\text{col}} + \epsilon) \quad (2.39)$$

$$\mathcal{H}_{\text{row}}(A) = -\frac{1}{m} \sum_{j=1}^m \sum_{i=1}^n p_{ji}^{\text{row}} \log(p_{ji}^{\text{row}} + \epsilon), \quad (2.40)$$

$$\mathcal{L}_{\text{entropy}} = \mathcal{H}_{\text{col}}(A) + \mathcal{H}_{\text{row}}(A) \quad (2.41)$$

With the entropy terms, each edge is encouraged to focus on a smaller, more confident set of inputs. That way, activation patterns that are diffuse or ambiguous get reduced[4].

METHODS

3.1 DATA GENERATION AND PREPARATION

We use the python package `pykan` developed by the authors of [15] to construct KANs. It uses the PyTorch library.

3.1.1 Data Generation with *pykan*-package

In `pykan` an internal synthetic data generation function `create_dataset` is provided. The function receives many variables. Notable ones are:

- `f`: the underlying formula to create the dataset
- `n_var`: number of variables, ideally similar to the amount of variables defined with f
- `train_num, test_num`: number of training and test samples.
- `ranges`: the range of the input variables. This can be adjusted for each variable in form of an array with shape $(n_var, 2)$. For a function $f(x, y)$ with two parameters it would look like

$$[[x_{min}, x_{max}], [y_{min}, y_{max}]].$$

Here $[x_{min}, x_{max}]$ denotes the interval over which data points for x are generated. If the interval is supposed to be the same for each input variable, only one has to be specified. The default is $[-1, 1]$ for all variables.

The `create_dataset`-function returns a dataset in form of a dictionary. The dictionary has entries for `train_inputs`, `train_labels`, `test_inputs`, and `test_labels`, based on the number specified with `train_num` and `test_num`. This serves as a train-test-split.

3.1.1.1 Normalization in *pykan*

Additionally, there is an option to normalize both input and label data using a standard z-score method

$$z_i = \frac{x_i - \mu}{\sigma}, \quad (3.1)$$

where z_i is the normalized value of data point x_i . Both μ and σ are the mean and standard deviation of the dataset respectively. This is done both

for input and for label data. Mean μ and standard deviation σ are calculated with PyTorch:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i, \quad (3.2)$$

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu)^2}. \quad (3.3)$$

N is the total amount of data points.

3.1.2 Creating Datasets for Synthesized Data

The dataset can also be generated with methods that differ from the `create_dataset`-function, as long as they have the same structure. We developed a method to generate datasets which better resemble the displacement field of mode III of the Williams series.

The displacement field of mode III is introduced in Eq. 2.4, and reads:

$$u_3^{(n)}(r, \theta) = \frac{c_n}{2\mu} r^{\frac{n}{2}} \tilde{H}_3^{(n)}(\theta), \quad \tilde{H}_3^{(n)}(\theta) = \begin{cases} 2 \sin \frac{n}{2} \theta & \text{for } n = 1, 3, \dots \\ 2 \cos \frac{n}{2} \theta & \text{for } n = 2, 4, \dots \end{cases} \quad (3.4)$$

Our models mainly investigate first term of the series

$$f = \sqrt{r} \sin \frac{\phi}{2}. \quad (3.5)$$

The material constants c_n and μ are disregarded for this thesis, since they do not influence the regression success of our models. (Note: In the results and discussion sections of this thesis the angle is denoted as ϕ , whereas the literature uses θ . They represent the same angle.)

We generate an input dataset for radius r , and angle ϕ , where the intervals are radius $r \in [0.01, 10]$ and $\phi \in [0, 4\pi]$. The data is evenly distributed, and 40,000 data points are created this way. Next, we apply function f on the input data to create a set of corresponding labels.

Afterwards both input and label data is normalized with the z-score method (see Eq. 3.1). Z-Score normalization results in a distribution around 0 with positive and negative values. However, in the odd terms of f , r appears under a square root. Thus r -values are shifted, so that all values for $r \geq 0$. This is done to prevent negative values underneath the square root.

The last step of our data generation is to split our data in a training and test set. To achieve that, 70% of our data is selected as training data, whilst the other 30% is chosen as test data. This selection is done randomly.

3.2 KAN

3.2.1 Initializing a KAN model

KANs are implemented using the pykan package. The following hyperparameters are the ones most relevant for initializing the first model.

- `width` denotes the architecture of our KAN model. For purely additive KANs a width entry like `[2, 4, 1]` represents two input nodes, four additive nodes, and one output node. If additional multiplicative operations are included in the hidden layer, the width is specified as `[2, [4, 3], 1]`. The second entry denotes four additive nodes and three multiplicative nodes (see Eq. 2.30).
- `grid` represents the number of grid intervals used to initialize the splines.
- `k` denotes the order of the piecewise polynomial of the splines. Default is `k = 3`, corresponding to a cubic spline.
- `seed` represents the seed used to ensure reproducibility. Ideally, formulas should be recoverable without relying on a specific initialization or seed, since these are not known beforehand for undiscovered formulas. Therefore, we use a random seed between 0 and 10,000 for most experiments.
- `noise_scale` is used to initially inject noise into the spline. In most experiments this value is set to 0.0, which helps to obtain a stable model.
- `grid_eps` denotes the partition of the grid. When `grid_eps = 1`, the grid is partitioned uniformly. At `grid_eps = 0` every grid interval has the same amount of data points, which results in a denser grid in regions of high data concentration. Values for `grid_eps` between 0 and 1 interpolate both strategies.
- `base_fun` specifies the base function $b(x)$ (see Eq 2.31). By default, the function $\text{SiLU}(x) = \frac{x}{1+e^{-x}}$ is used.
- `scale_base_mu` and `scale_base_sigma` are used to initialize the scale factor w_b of the base function $b(x)$: $w_b \sim \mathcal{N}(\mu, \sigma^2)$. w_b is sampled from a normal distribution with mean μ and standard deviation σ for each edge individually.

The `refine`-function is used to extend the grid size, which creates a new KAN model with a refined grid. Each parameter is inherited from the original initialization, except for `grid`, which is passed a finer grid size (see section 2.5.1.3). The different grids are given as a list, like `[grid1, grid2, grid3]`, where `grid1` is used to initialize the KAN model, and `grid2` and `grid3` are used in the `refine`-function

3.2.2 Training a KAN model

The `fit`-function is used to train the KAN model. The function is executed after the KAN initialization and every following refining step. One complete training phase is also called an epoch.

The following hyperparameters are important for the training of our models.

- `dataset` is the dataset we generated with one of the methods explained in sections 3.1.1 and 3.1.2.
- `opt` denotes the optimizer used. We use LBFGS.
- `steps` specifies the number of training steps the algorithm performs during each training epoch. In this thesis the standard is 200.
- `lamb` represents λ_{reg} , the overall regularization weight.
- `lamb_l1` denotes λ_{L1} , the regularization weight for L1-regularization.
- `lamb_entropy` specifies $\lambda_{entropy}$, the regularization weight for entropy regularization.
- `lamb_coef` represents the regularisation weight for regularization on spline coefficient.
- `lamb_coefdiff` denotes the regularization weight for the regularization of the difference of nearby coefficients.
- `lr` indicates the learning rate of the model.

During training the training and test losses are saved to monitor the training process. After each training epoch the model is refined, and trained again on the new grid size. The model is trained minimizing the objective value, which is the Root Mean Square Error (RMSE) (see Eq. 2.3.1.1) added to the regularization term $\mathcal{L}_{\nabla\} \}$ (see Eq. 2.33):

$$\mathcal{L}_{\lceil\} = \mathcal{L}_{RMSE} + \mathcal{L}_{\nabla\} \}, \text{ with } \mathcal{L}_{RMSE} = \sqrt{\mathcal{L}_{MSE}} \quad (3.6)$$

3.2.3 Symbolic Recovery

Two different methods for symbolic recovery are included in the `pykan` package. Both fix the edges of a KAN to a function from a predefined library, that is also included in the `pykan` package.

`fix_symbolic` allows us to specify which edge should be assigned a particular function. An edge is identified by the layer `l`, input neuron `i`, and output neuron `j`. The function assigned to the edge is specified by the `fun_name` parameter. This allows the use of prior knowledge for symbolic recovery.

`auto_symbolic` on the other hand determines the function assigned to each edge automatically. The selection is based on a criterion that combines R^2 score with a library-specific complexity measure (see section 3.2.6). A different library of candidate functions can be passed via the `lib` parameter. The final selection is based on minimizing a loss function $\mathcal{L}_{sel}$ that combines a complexity given by a library (see Sec. 3.2.6), and the R^2 score of the corresponding analytical function:

$$\mathcal{L}_{sel} = w_c * c_{lib} + (1 - w_c) * R^2, \quad (3.7)$$

where w_c is the weight that is attributed to the complexity value in the library.

3.2.4 Pruning

After each training, before refinement, the model is pruned with the function `prune`. The algorithm calculates an attribution score for each edge and each node. Edges and nodes with an attribution score below a predefined threshold are removed from the model (see Fig. 3.1). Attribution scores are computed by propagating output importance backward through the network, resulting in relevance scores for nodes and edges. The default values are `node_th` = 10^{-2} for nodes, and `edge_th` = $3 \cdot 10^{-2}$ for edges. Pruning is applied to obtain a sparse model, which improves interpretability and reduces the number of terms in the symbolically recovered formula.

3.2.5 Plotting Model Structure

For visualization of the model architecture after training steps, the current model structure is plotted using the `plot` function. Example figures and the pruning mechanism are shown in Fig. 3.1. Symbolically fixed edges are colored red. We can see that after the first training, the attribution score is below the pruning threshold for many edges/ nodes. Therefore, pruning results in a sparse structure with only three remaining edges. As a last step the edges are fixed symbolically, which replaces spline-based activation functions with smooth analytical functions.

3.2.5.1 Optuna - Hyperparameter optimization

For hyperparameter optimizations, the framework “Optuna” [1] was used. Optuna organizes experiments in `studies`, where each run is called a `trial`. In each `trial`, hyperparameters are selected from a predefined search space. The objective value serves as evaluation metric. In our studies, the objective was to minimize the RMSE of the model after symbolic recovery.

The studies were done on a synthetically generated data set (see section 3.1.2).

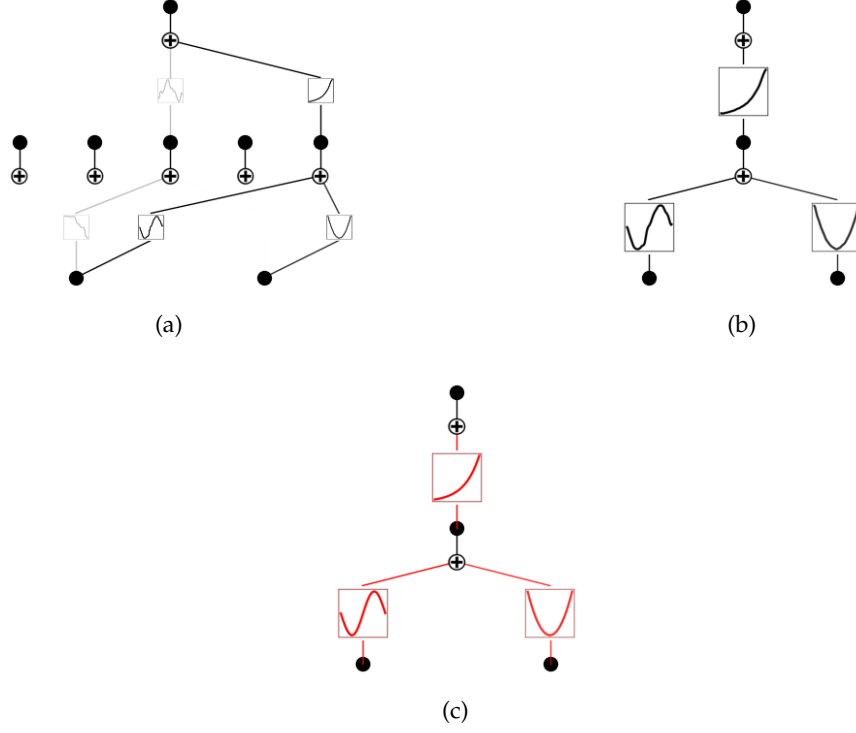


Figure 3.1: (a) KAN model with architecture [2, 5, 1] after training; (b) KAN model after pruning; (c) KAN model after symbolic fixation

Each trial consisted of training a KAN model with increasing grid sizes [50, 100, 200]. For each grid, size the model trained for 200 training steps. After grid size 200 the model was fixed symbolically, followed by an additional training. The RMSE resulting from the last training was used as objective value.

We conducted several optimization studies for different configurations.

STUDY A: REGULARIZATION HYPERPARAMETERS FIRST TERM In Study A we optimized in 350 trials for the following regularisation hyperparameters:

- $\lambda_{L1} \in (10^{-10}, 1)$, logarithmic scale
- $\lambda_{\text{entropy}} \in (0, 5)$
- $\lambda_{\text{coef}} \in (10^{-10}, 1)$, logarithmic scale
- $\lambda_{\text{coefdif}} \in (10^{-10}, 1)$, logarithmic scale
- $\text{grid_eps} \in (0, 1)$
- $\text{lr} \in (10^{-5}, 0.1)$, logarithmic scale.

The study was conducted on data generated from the underlying function $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$, which represents the first term of the displacement

field in mode III (see Eq. 2.4). The KAN architecture was given as `kan_width` = [2, [0, 1], 1], which is the optimal architecture for a KAN of depth 2.

STUDY B: REGULARIZATION HYPERPARAMETERS MULTIPLE TERMS In Study B, the following regularisation hyperparameters were optimized over 200 trials:

- $\lambda_{L1} \in (0, 0.03)$
- $\lambda_{\text{entropy}} \in (0, 0.1)$
- $\lambda_{\text{coef}} \in (0, 0.005)$
- $\lambda_{\text{coefdif}} \in (0, 0.01)$

The study was conducted on data generated from the underlying function $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi)$, which represents the first two terms of the displacement field in mode III (see Eq. 2.4). The learning rate $\text{lr} = 10^{-3}$. The KAN architecture was given as `kan_width` = [2, [5, 5], [5, 5], 1].

STUDY C: OPTIMIZING KAN ARCHITECTURE AND PRUNING THRESHOLD Lastly, we investigated whether KANs can find the optimal structure. Depth, addition nodes, and multiplication nodes were tested, as well as the pruning threshold for nodes. As a last layer, the structure of one multiplication node, and no addition nodes ([0, 1]) was predefined. In Study C we optimized in 250 trials for the following hyperparameters:

- `width_add` $\in [0, 1, 2, 3, 4]$
- `width_mult` $\in [0, 1, 2, 3, 4]$
- `depth` $\in [1, 2, 3]$
- `node_th` $\in (0.01, 0.2)$

For `depth` = 1 this means the architecture was [2, [0,1]] Again, the study was conducted on data generated from the underlying function $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$. The following values were passed for regularization parameters:

Parameter	Value
λ_{reg}	$3 \cdot 10^{-5}$
λ_{L1}	10^{-4}
λ_{entropy}	10^{-4}
$\lambda_{\text{coefficient}}$	10^{-3}
$\lambda_{\text{coefficient difference}}$	10^{-4}
Learning rate	10^{-3}

Table 3.1: given regularization parameters for Study C

3.2.6 Complexity of Symbolic Recovery Expressions

For symbolic recovery the library given in the pykan package is used. Each analytic function recorded in the library is assigned a “complexity” measure. Traditionally, complexity calculations of symbolically recovered functions are dependent on the number of functions that contribute to the final expression, which represents the “function complexity”. On top of that, the number of active edges used in the final model can be interpreted as “KAN complexity”. Therefore, we introduce three different complexity values:

- c_{lib} : complexity value assigned to an individual analytic function that is symbolically fixed on an edge of the network
- c_{func} : total complexity of the recovered expression, computed as the sum of the library complexities c_{lib} of each contributing analytic function.
- c_{kan} : structural complexity of the model, defined as the sum of complexity values c_{lib} associated with the active edges in the final model after symbolic recovery:

$$c_{\text{kan}} = \sum_{i=1}^n c_{\text{lib},i}$$

, where n denotes the number of active edges.

3.2.6.1 Pareto Fronts

To analyze the trade-off between complexity and the model performance we utilize Pareto fronts. A solution is part of the Pareto front if there exists no other solution to the objective which is better in both objectives. This allows identifying models which offer the best balance of complexity vs. performance [24, p. 286].

In our case we plot Pareto fronts of two different libraries: One with the implementation used in the pykan-package where many analytical functions had a complexity term c_{lib} assigned to them, and another library where every value for c_{lib} was set to “1”. The libraries can be found in Appendix A.

Basis for the Pareto fronts was data generated on the underlying formula

$$f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right), \quad (3.8)$$

as well as the following hyperparameters:

- $\lambda_{\text{reg}} = 0.01$
- $\lambda_{L1} = 1$
- $\lambda_{\text{entropy}} = 2$

- $\text{lr} = 0.01$.

RESULTS

4.1 KAN TRAINING EXAMPLES

To investigate the suitability of KANs as an algorithm for symbolic regression tasks in context of the Williams expansion, prior experiments are conducted. They are used to assess the potential capabilities and obstacles for model training.

4.1.1 Symbolification for an Example KAN

Firstly, the initialization, training and symbolification of a KAN will be analyzed, that was used as an example in the publication by Liu et al. [15].

The dataset is generated by the process described in Sec. 3.1.1 using the function

$$f(x, y) = \exp(\sin(\pi x) + y^2) \quad (4.1)$$

The KAN is initialized according to Section 3.2.1 with

- width = [2,5,1]
- grid = 3
- k = 3
- seed = 0.

The resulting KAN initialization is shown in Fig 4.1.

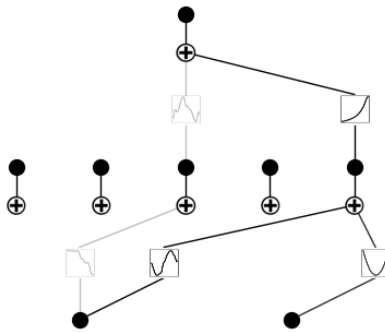


Figure 4.1: The initialized KAN with three layers: the input layers with two input nodes representing x , and y , five hidden addition nodes, and one output node.

The training commences in epochs as described in Section 3.2.2, with each epoch training on a finer grid. In Fig. 4.2 the training and symbolification

phases are visualized and compared. For symbolic regression tasks, subsequent to the model training a symbolification step is performed, with a final training done on the symbolically fixed model.

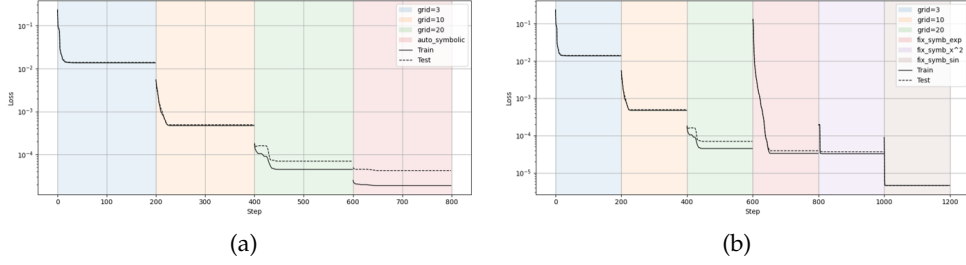


Figure 4.2: Training of a KAN with grid sizes = [3, 10, 20]. In (a) the symbolification was done with the `auto_symbolic` - function of the `pykan` package, whereas (b) symbolically fixes the edges manually using `fix_symbolic` (see Sec. 3.2.3). The resulting RMSE-loss is noticeably different, with (b) resulting in a lower training error.

The output of the KANs can be expressed as a formula. For the automatic symbolification, where analytic functions are assigned to edges according to the criterion defined in Eq. 3.7, the resulting formula is given in 4.2, while the output of symbolic recovery with manually fixed edges is shown 4.3. The KAN using `auto_symbolic` to recover the underlying data did not manage to map to the desired function. By incorporation prior knowledge about the data distribution and generation, the correct functions could be assigned to the edges for the manually configured symbolic recovery.

$$f(x, y) \approx 182.13 \tanh(0.48y^2 + 0.51 \sin(3.14x) - 2.95) + 182.14 \quad (4.2)$$

$$f(x, y) \approx 1.0e^{1.0 \sin(3.14x) + 0.97y^2}. \quad (4.3)$$

The selection criterion for automatic symbolic recovery is visualised in Fig. 4.3. The underlying formula in Eq. 3.7 defines a loss function $\mathcal{L}_{sel}$ for each candidate function in the assigned library, and selects the function with minimal loss. Since the loss value for *tanh* was minimal, the computed loss function $\mathcal{L}_{sel}$ selects *tanh* as the appropriate function to symbolically fix the corresponding edge. This selection is performed for each edge.

The data distribution for training and test data used for our KAN models is shown in Fig. 4.4, while the error distribution is shown in Fig. 4.5. The error distribution uses a calculated error for each data point given as

$$e_{dist} = |y_i - \hat{f}(r_i, \phi_i)|. \quad (4.4)$$

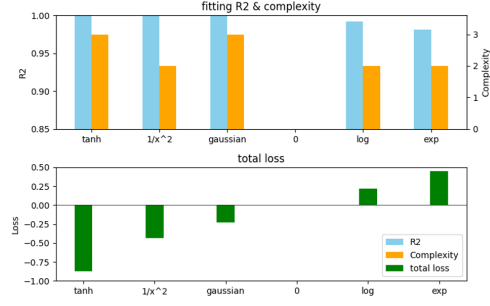


Figure 4.3: Metrics showing parameters R^2 and complexity for the evaluating function $\mathcal{L}_{sel}$ as well as the resulting loss values.

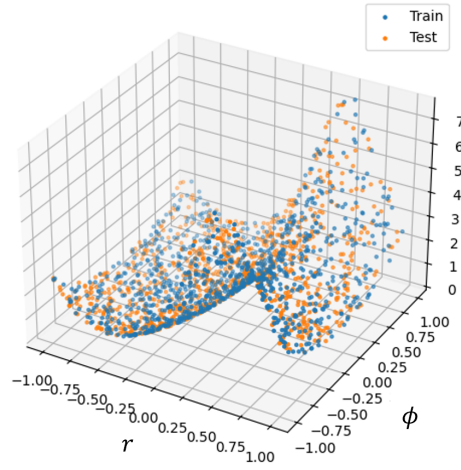


Figure 4.4: Data distribution for both examined KAN models. Training data is plotted together with test data. The same data was used for both experiments.

Here y_i represents the original labels. The labels y_i are compared to the prediction $\hat{f}$ calculated using the corresponding input data (r_i, ϕ_i) . The error distribution for the model using automatic symbolic fixation shows higher absolute error values in certain regions, by up to two orders of magnitude.

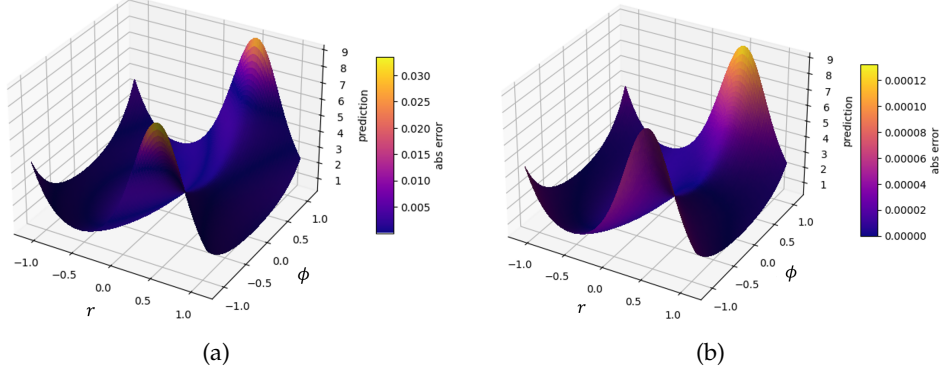


Figure 4.5: Plots of the absolute error $|y_i - \hat{f}(r_i, \phi_i)|$ of predictions $\hat{f}(r_i, \phi_i)$ with regards to the labels y_i . Plot (a) corresponds to the model where `auto_symbolic` was used, plot (b) shows the error distribution for the model with manually fixed edges.

4.1.2 KAN behaviour near singularities

As introduced in Eq. 2.2, the first term ($n = 1$) for stress fields $\sigma_{ij}(r, \theta)$ is dependent on $\frac{1}{\sqrt{r}}$. For $r \rightarrow 0$ the term exhibits singular behaviour. NNs are known to approximate continuous functions on compact domains (see Sec. 2.3.2, [12, p. 66]). However, functions with singularities are not continuous at the singularity, while the gradients of the function can become arbitrarily large near the singular point. Therefore, we investigate a function with singularity to examine how KANs behave in the vicinity of a singularity.

For the experiments we consider a function

$$f(x) = \frac{1}{x} \quad (4.5)$$

Two datasets are generated, which consist of input variables x and y , while but the output function only depends on x . The datasets are created with the method explained in Sec. 3.1.1.

The input variables of the first data set are set to be of the interval $(x, y) \in [0.1, 1]$. For the second experiments, data is generated closer to the singularity at $(x, y) \in [10^{-4}, 1]$. The KANs are initialized with width = [2, 1, 1].

In Fig. 4.6 we visualize both logarithmic (left) and absolute (error) error distributions. The absolute error is calculated according to Eq. 4.4, whereas the logarithmic error is the result of

$$e_{\log} = \ln(e_{\text{dist}}). \quad (4.6)$$

The logarithmic error is used to visualize changes in values that are too small to be clearly visible with the absolute error.

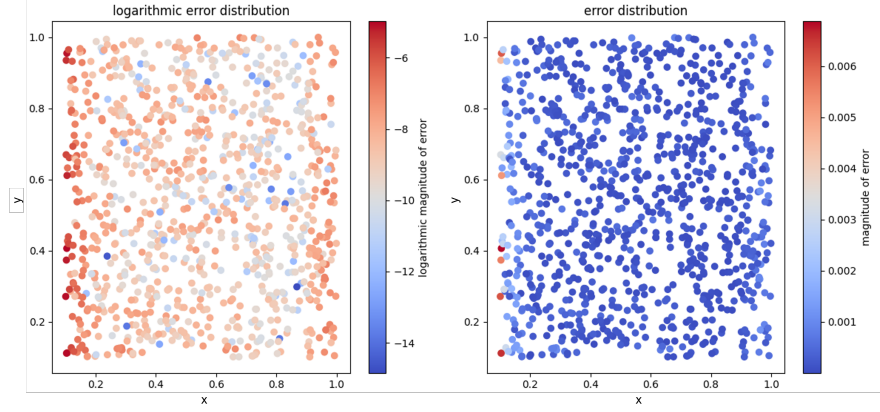


Figure 4.6: scatter plot of logarithmic error (left) and absolute error (right) for $(x, y) \in [0.1, 1]$

Comparing the error distribution in Fig. 4.6 with the error plots in Fig. 4.7, both the absolute error scale and logarithmic error scale are higher in Fig. 4.7 that depicts the error distributions for $x, y \in [10^{-4}, 1]$. In the logarithmic error plot in Fig. 4.7 the highest error values appear near the lower bound of to $x \rightarrow 10^{-4}$ with the error values becoming noticeably smaller when the data points are further away from the values where a significant gradient is expected. In comparison Fig. 4.6 has a more equally distributed error landscape, with notable spikes close to $x = 0.01$.

Analyzing the train-test-plots in Fig. 4.8 the training for $x \in [10^{-4}, 1]$ resulted in a model, where training and test losses diverged from each other by an order of magnitude. The training of the model for $x \in [0.1, 1]$ was more successful in that regard, as both training and test error converged and reached smaller RMSE values. Still with higher grid sizes, some difficulties in the training are visible, which may indicate difficulties in learning regions with large gradients in intervals close to $x = 0.1$.

The model plots 4.9 depict the KAN architecture after training and grid refinement. For $(x, y) \in [0.1, 1]$ (a) the majority of activations are on the edge that connects to the input node of x . The model corresponding to $(x, y) \in [10^{-4}, 1]$ shows a significant contribution of the input variable y to the final output. Since the target function does not depend on y , this indicates that the model incorporates a non-contributing input variable in the resulting output. This suggests that the model utilizes the additional input dimension to approximate the target function in regions with large gradients.

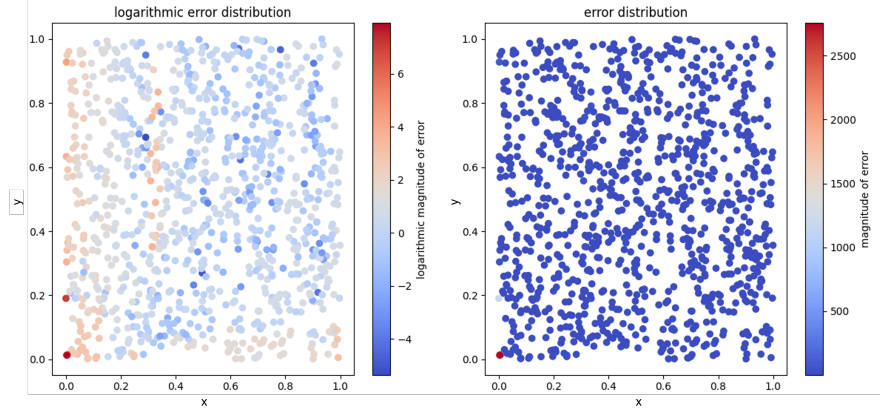


Figure 4.7: scatter plot of logarithmic error (left) and absolute error (right) for $(x, y) \in [10^{-4}, 1]$

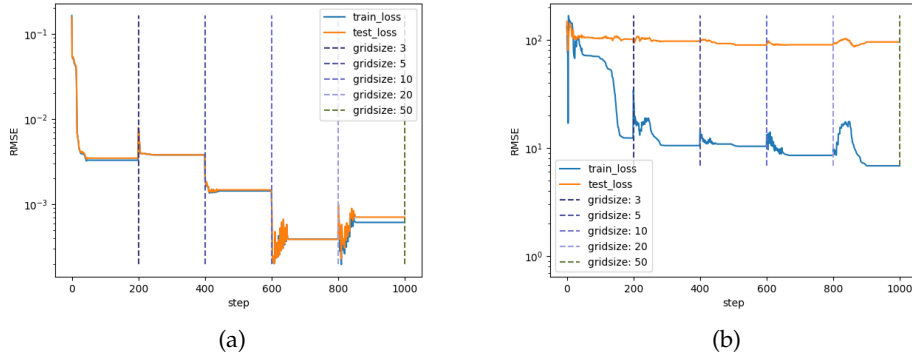


Figure 4.8: These plots illustrate training and test losses. For $(x, y) \in [0.1, 1]$ (a) the losses remain close, while for $(x, y) \in [10^{-4}, 1]$ training and test losses converge

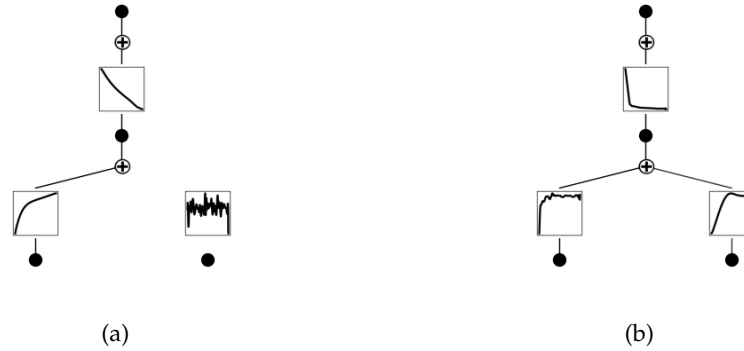


Figure 4.9: Comparison between the activation functions in the model plots for a singularity. (a) The model for $(x, y) \in [0.1, 1]$ mainly utilizes one edge, whereas the model plot (b) for $(x, y) \in [10^{-4}, 1]$ shows that both x and y contribute significantly to the output.

4.2 KANS FOR FIRST TERM DISPLACEMENT FIELD MODE III

The infinite series for mode III displacement field is given in equation 2.4. For $n = 1$ the term reduces to

$$f(r, \theta) = \frac{c_n}{2\mu} r^{1/2} \sin\left(\frac{\theta}{2}\right). \quad (4.7)$$

With c_n and μ being material-dependent variables, and since we do not investigate specific materials, the exact values for the training process are treated as constants. We set $\frac{c_n}{2\mu} = 3$. This formula will serve as basis for all experiments if not stated otherwise. The KAN is initialized with a grid size of 50, which is subsequently increased to 100 and 200 (see Fig. 4.11).

The minimal architecture of a KAN required to represent Eq. 4.7 fully is $\text{kan_width} = [2, [0, 1]]$. The model represents two input variables, two additive subnodes, that get multiplied in the subsequent multiplication layer (see Fig 2.6). Since the last operation is a single multiplication, the multiplication node can be used as the output node for the whole network. However, this architecture is not used in this case.

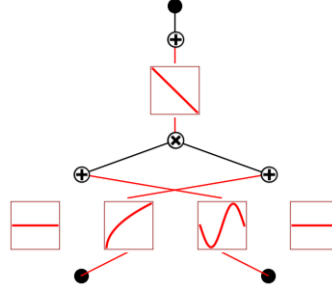


Figure 4.10: model plot after symbolification of trained edges, displacement field mode III

Fig. 4.10 shows an additional layer with one edge. In that layer, the recovered formula is linear, which only additive or multiplicative constants. Therefore, the structure of the recovered expression stays largely the same. In this case the final architecture of the KAN is $[2, [0, 1], 1]$.

The recovered formula by this KAN is

$$f(r, \phi) \approx \left(-0.3744\sqrt{10r + 0.034} \right) \left(0.91 \sin(1.8\phi) \right).$$

Additive constants below the threshold of 10^{-6} were removed manually for clarity. This was done for readability purposes only. If the KAN was trained on real experimental data, removing those terms would require justification, since there are universal physical constants of values for below the threshold of 10^{-6} (e.g. Planck constant $h \approx 6.626 \cdot 10^{-34}$, and it would remain unclear whether those terms indicate meaningful physical effects).

By analyzing the training of the KAN which is shown in Fig. 2.6, we can see that the training converges to an RMSE of approximately 10^{-2} . However, since the symbolic fixing was able to find the correct functions for each edge, the model performance in regards to the training and test losses is increased by about four orders of magnitude (see Fig. 4.11). In Sec. 4.5 we will see that a final loss value in range of $10^{-6} - 10^{-7}$ can indicate successful symbolic regression for data generated with Eq. 4.7. However, the goal is not necessarily to manage the best loss value, but to find the correct underlying formula for the data. According to the results presented in this chapter, both seem to correlate heavily, but this relationship is not guaranteed in general.

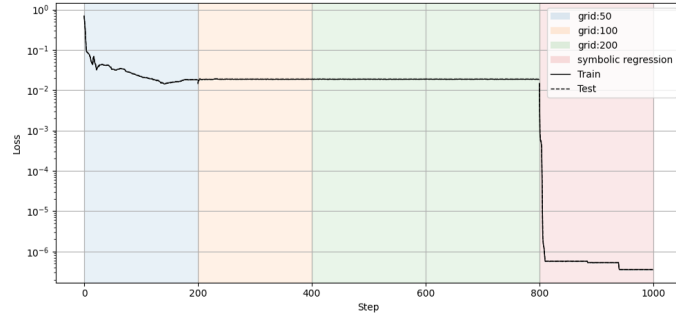


Figure 4.11: train-test error of successful symbolic regression for the first term of displacement field for mode III

In Fig. 4.12 the absolute error in regards to the distribution of the generated dataset is shown. The absolute error is low with the maximal deviations being about 10^{-6} . Still, a clear pattern is recognizable regarding the relationship of negative normalized ϕ -values and the absolute error: the error is significantly bigger $\phi < 0$.

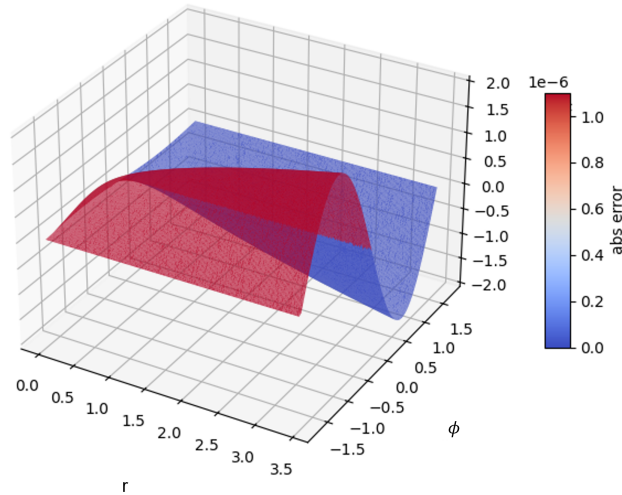


Figure 4.12: absolute error of successful symbolic regression first term mode III

Overall it can be observed that the recovered formula has the same structure as the formula that was used to create the dataset: r is part of the radicand, and ϕ is part of a trigonometric function (in this case sine). Still, the exact formula was not recovered in the same way as in our example KAN with manual fixing of the edges (see 4.1.1). A factor is that the data was normalized, but afterwards no “denormalization” was applied. The goal for the first term of mode III was to determine whether a KAN can represent the structure of the underlying generated formula successfully. In that regard, the results indicate that this experiment was successful.

Since the following section investigates hyperparameters and their influence, some relevant hyperparameters used in this experiment are shown here:

Parameter	Value
λ_{reg}	0.01
λ_{L1}	1
λ_{entropy}	2
Learning rate	0.01

Table 4.1: hyperparameters for successful SR of first williams term of displacement field in mode III

4.3 HYPERPARAMETER OPTIMIZATION

We showed that KANs are capable of recovering the structurally underlying formula from a given dataset, at least for the first term of the Williams series for mode III displacement. The next step is to optimize hyperparameters, in particular those responsible for regularization. This is done to increase the performance of the model.

4.3.1 Hyperparameter Optimization for First Term Mode III Displacement Field

Study A

The first Optuna study is concerned with optimizing regularization parameters for the first Williams term of the displacement field in mode III, which is given as

$$f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right).$$

Since there are many hyperparameters we limit the search space. In this experiment, the focus lies on regularization. As we can see in the regularization equation 2.33, $\mathcal{L}_{\text{reg}}$ is dependent on λ_{reg} , λ_{L1} , λ_{entropy} , λ_{coef} and $\lambda_{\text{coefdiff}}$. λ_{reg} influences all other regularization parameters. As they are not independent from each other, we have to decide, whether to optimize λ_{reg} , but keep the ratio between the other λ 's constant, or fix λ_{reg} and optimize for the sub-

regularization hyperparameters. In this thesis, the latter was chosen. For this particular study, λ_{reg} was set to 0.315.

For Optuna study A (see Sec. 3.2.5.1) depicted in Fig. 4.13, the best trial had the results shown in Tab. 4.2. Notable are the small values for the learn-rate, and the regularization factor λ_{L1} .

Parameter	Value
λ_{L1}	1.746632795283256e-08
$\lambda_{entropy}$	0.11461087041409745
$\lambda_{coefficient}$	0.00011233764052417883
$\lambda_{coefficient\ difference}$	0.0005635530234120837
Learnrate	1.4261967677419512e-05
grid_eps	0.5329905951166182

Table 4.2: optimized hyperparameters for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$

The objective value that was minimised by the optimization was the RMSE. For the parallel coordinate plot, the darker the graph, the closer is the objective value calculated by that trial. When looking at the parallel coordinate plot which depicts combinations of tested parameter values, we can see that no apparent pattern emerges. A lack of clear structure in the plot can indicate that no strong trends for the hyperparameters were identified. Only a tendency to lower $\lambda_{entropy}$ is visible, but the results are far from conclusive. Still, this is pointing to $\lambda_{entropy}$ being the most important hyperparameter for this study, which is confirmed by the bar plot for parameter importance.

The KANs were initialized with different kan_widths, similar to Sec. 4.3.3. The grid sizes were [50,100,200].

4.3.2 Hyperparameter Optimization for Higher Terms of Mode III Displacement Field

As already established, the williams series is an infinite series. Therefore, we want to expand our hyperparameter search to cases with several additive terms with Study B (see Sec. 3.2.5.1).

As a starting point we choose

$$f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi),$$

which are just the first two terms of the Williams expansion for the displacement field mode III. The study ran for 200 trials, the initialized KAN

¹ Fig. 4.13, Fig. 4.14 Fig. 4.16 all represent comparable plots: upper left: objective values for each trial including the pareto front. upper right: influence of each investigated parameter. bottom: plot maps each trial and their respective hyperparameter values to objective value

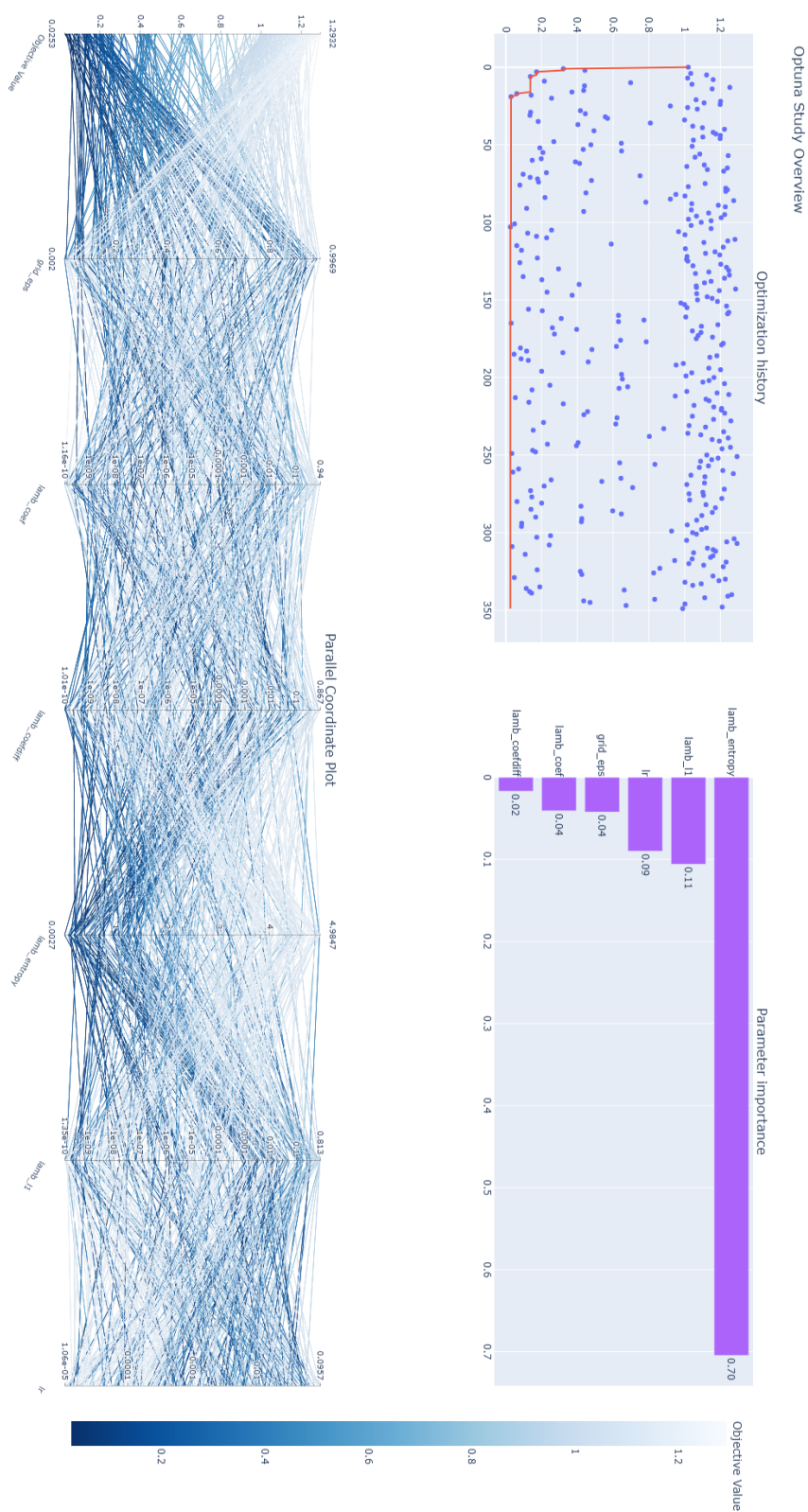


Figure 4.13: Optuna study on hyperparameters for the first Williams term of mode III displacement field.

architecture was $\text{kan_width} = [2, [5, 5], [5, 5], 1]$. Such a wide and deep network is chosen to see how well the KAN can find the optimal width for each layer, without directly giving the optimal architecture. A problem for the first optimization was the focus on the right architecture. This poses the risks of overfitting, which would make the results not applicable for more generalized use cases.

The other set hyperparameter were $\text{grids} = [20, 50, 100, 200]$, $\lambda_{\text{reg}} = 1$, $\text{learnrate} = 0.001$. The results are easier readable for $\lambda_{\text{reg}} = 1$, which is why this change was made. The optimal learnrate of the first study proved to be too small, which is why this study uses a higher one.

The best trial in this study had the hyperparameters in Tab. 4.3. That trial used a value several orders of magnitude higher for λ_{L1} than the best trial in the study for the first term did.

Parameter	Value
λ_{L1}	0.00830036568072633
λ_{entropy}	0.018403278011087997
λ_{coef}	0.004549770141639009
λ_{coefdif}	0.007910309457354069

Table 4.3: optimized hyperparameters for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi)$

The results of the Optuna study in Fig. 4.14 show, that λ_{entropy} has the largest influence, followed by λ_{L1} . However, the Parallel Coordinate Plot, does not reveal a clear trend towards the most successful trials.

Nevertheless, the training produced a sparse model.

The recovered formula can be found in B.1.

4.3.3 KAN Architecture Optimizing

In this section, we present Optuna Study C (see Sec. 3.2.5.1) regarding width and depth in addition to the threshold for pruning nodes, see Fig. 4.16. The focus lies on determining which configuration of addition and multiplication nodes results in the best performing models, and to analyze whether an Optuna-optimized KAN model can recover the optimal architecture for the specific underlying function. The function used to generate the data is $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$.

The results of the model performance depending on different depths is shown in Fig. 4.17. The best results correspond exclusively to $\text{depth} = 1$, which had the structure $[2, [0, 1]]$.

The results of the best trial are depicted in Table 4.4

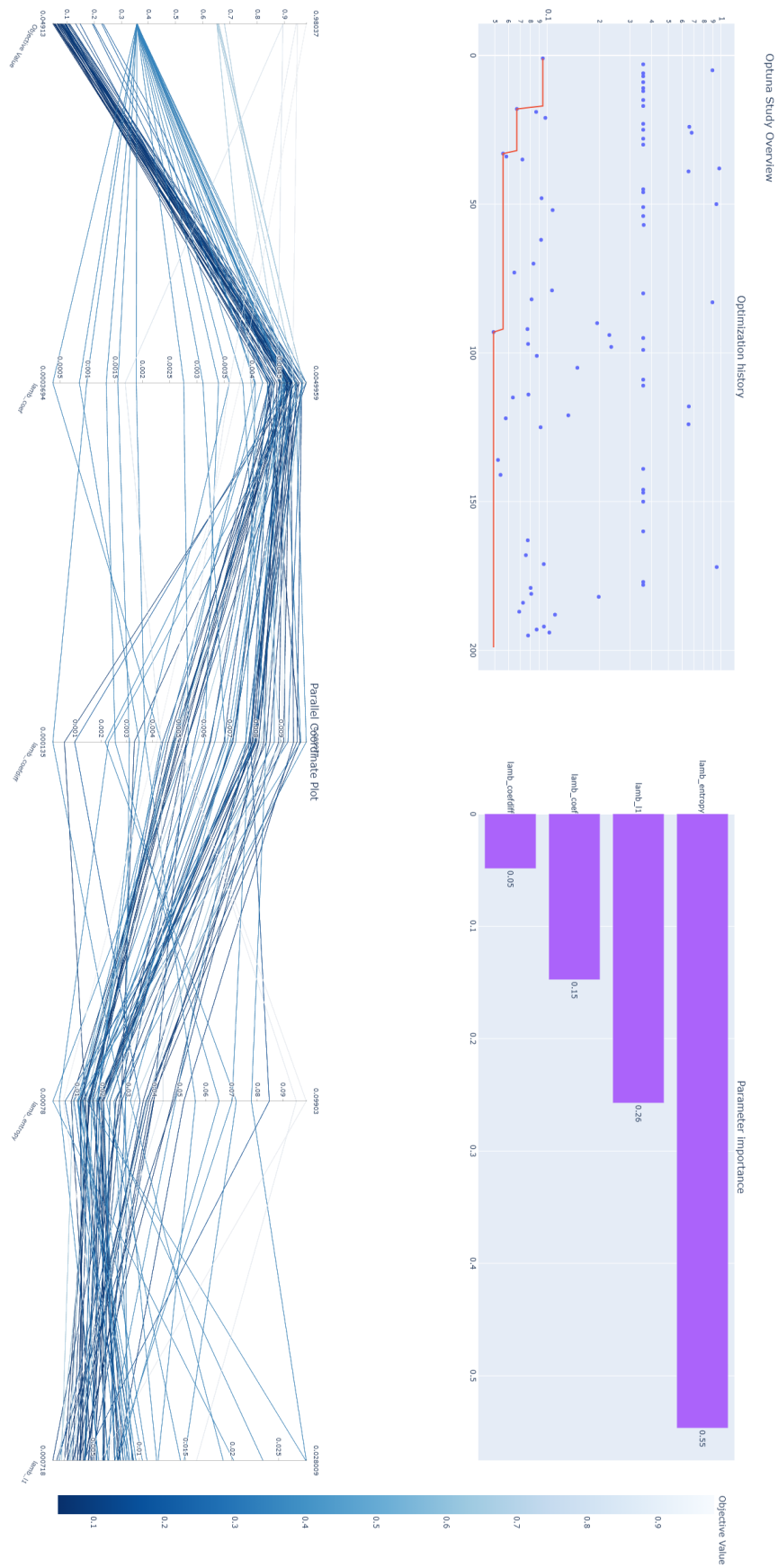


Figure 4.14: Optuna study of hyperparameters for first and second term.

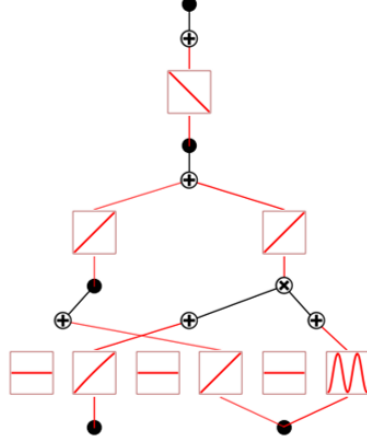


Figure 4.15: best trial model after symbolification in optuna study for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi)$. The plot shows several symbolically fixed linear functions, a trigonometric function was symbolically recovered for ϕ . The input from r is symbolically fixed as $\sqrt{r}$.

Parameter	Value
width_add	4
width_mult	0
depth	1
node_th	0.085

Table 4.4: optimized parameters for KAN architecture and pruning threshold

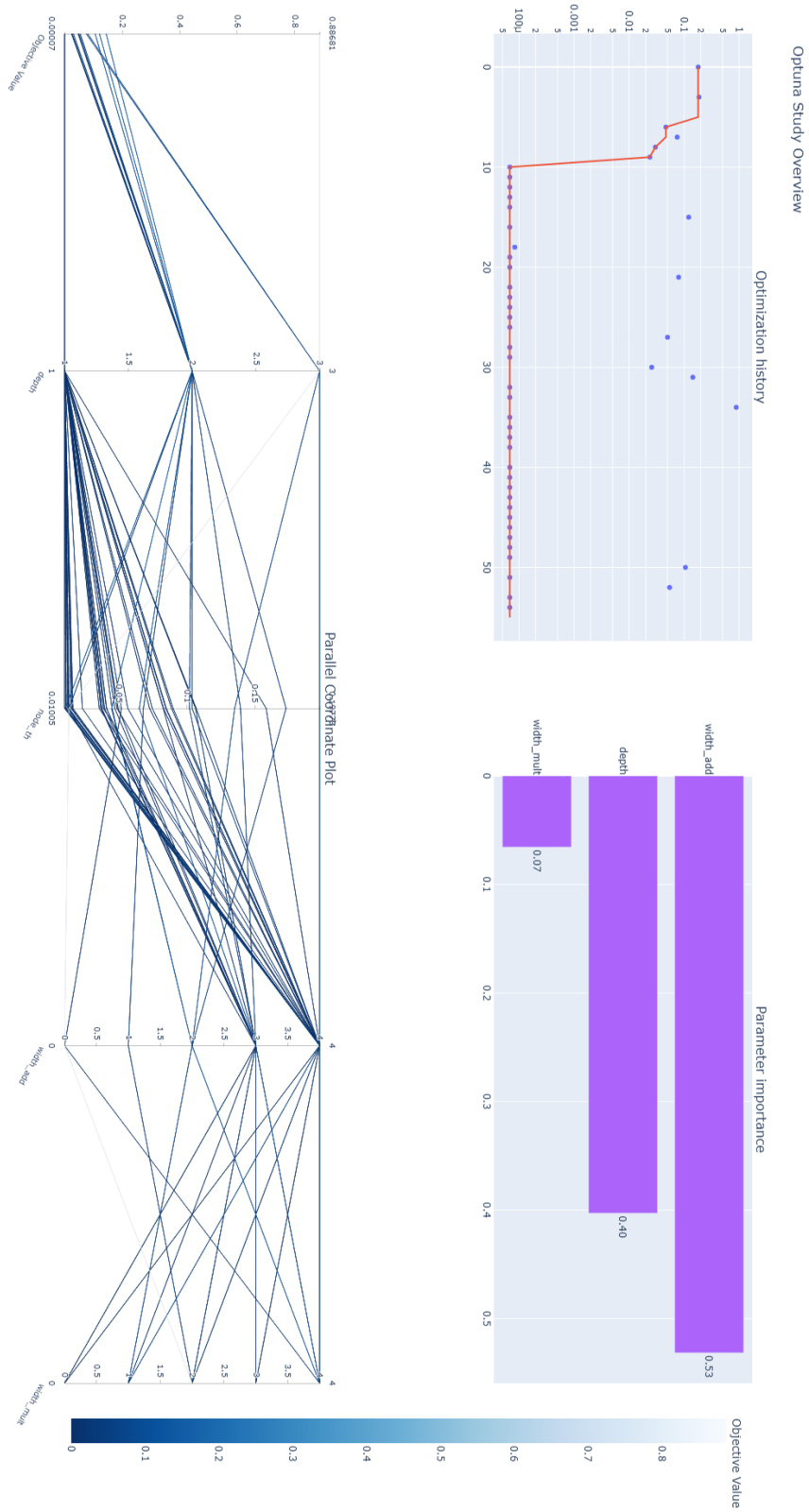


Figure 4.16: Optuna study of KAN architectures, specifically width and depth.

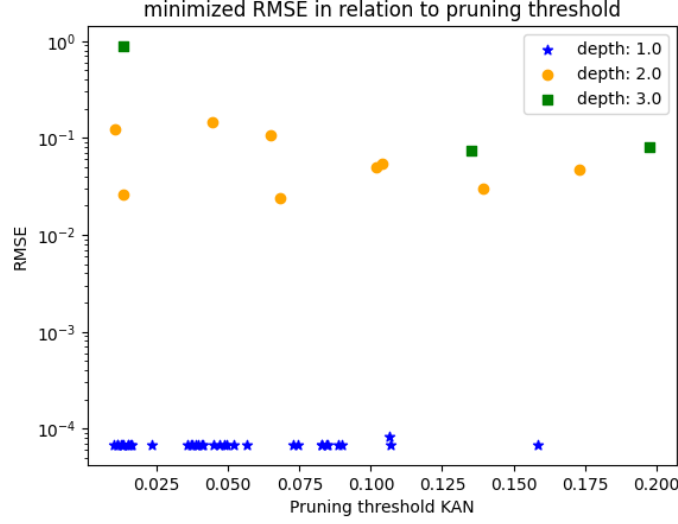


Figure 4.17: Models with depth 1 show significantly lower RMSEs compared to models with a higher number of layers for $f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right)$

4.3.4 Effect of Hyperparameters on Edge Usage in KANs

To determine the effect of different values of λ_{L1} and λ_{entropy} , a grid of resulting KAN models is presented in Fig. 4.18. The other input parameters were $\lambda_{\text{reg}} = 0.01$ and $lr = 0.01$. To acquire comparable results, $\text{seed} = 1$ was chosen. The model plots were visualized after the initial training, before pruning.

By comparing column $\lambda_{L1} = 0$ with the other columns, it can be observed, that higher values for $\lambda_{L1} = 0$ lead to sparser models. although some of the effect is compensated for by higher values of λ_{entropy} . On the other hand, the influence of λ_{entropy} is not as clearly visible, besides the focus on a fewer activation functions in column $\lambda_{L1} = 0$. For higher values of both λ_{L1} and λ_{entropy} , more edges are active.

In the following experiments, neither λ_{coef} nor λ_{coefdif} is considered. This choice is motivated by their default value of 0 in the pykan-package, their limited discussion in the original KAN publications, and their small values and contributions to the best trials in our optimization studies. Additionally, going forward $\lambda_{L1} = 1$ and $\lambda_{\text{entropy}} = 2$ are chosen.

4.4 HIGHER TERM EXPLORATION

In this section we explore the training of KANs with data that is generated based on higher level terms of the Williams series(see section 2.2.1). The focus lies on evaluating whether a KAN can recover the optimal architecture for higher terms with standard regularization parameters. The underlying formula is given by

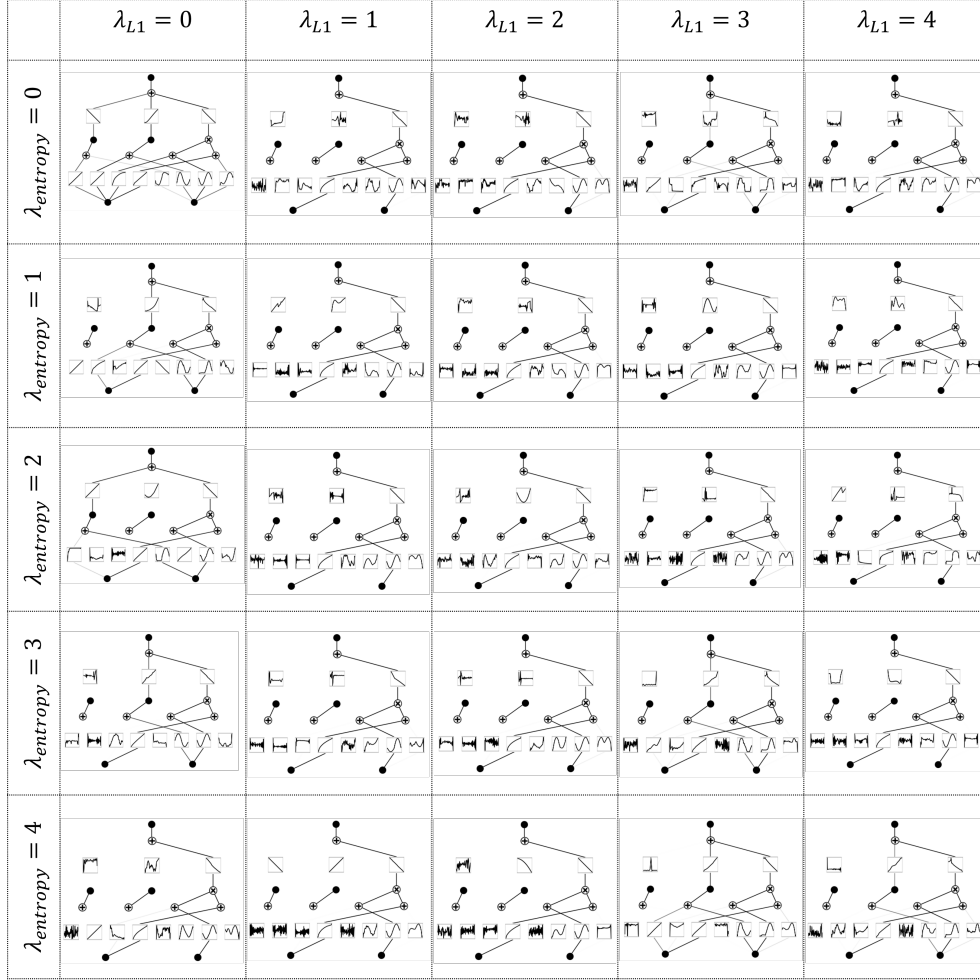


Figure 4.18: model plots before pruning with different values for λ_{L1} and $\lambda_{entropy}$, with KAN architecture `kan_width = [2, [2, 1], 1]`, and data generation function $f(r, \phi) = 3\sqrt{r} \sin \frac{\phi}{2}$

$$f(r, \phi) = r^{\frac{3}{2}} \sin\left(\frac{3}{2}\phi\right) + \sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi). \quad (4.8)$$

The following hyperparameters were used:

- $\text{kan_width} = [2, [5, 5], 1]$
- $\lambda_{\text{reg}} = 0.01$
- $\lambda_{L_1} = 1$
- $\lambda_{\text{entropy}} = 2$
- $\text{lr} = 0.01$.

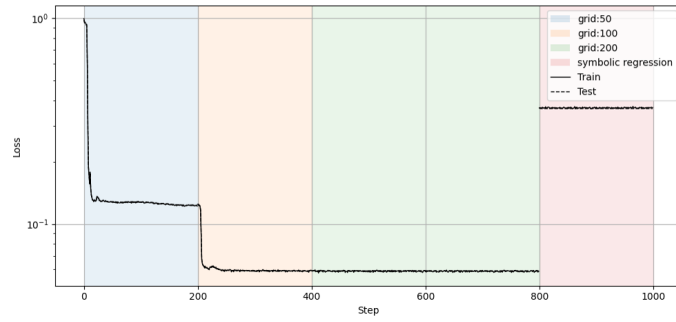


Figure 4.19: training and test loss for higher terms $f(r, \phi) = r^{\frac{3}{2}} \sin\left(\frac{3}{2}\phi\right) + \sqrt{r} \sin\left(\frac{\phi}{2}\right) + r \cos(\phi)$

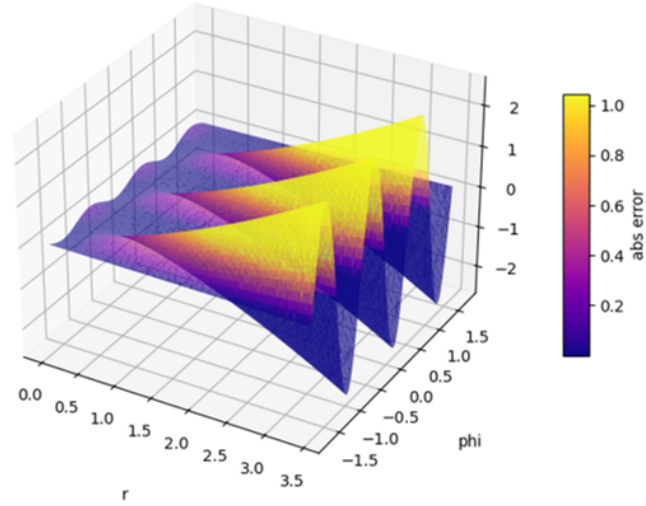


Figure 4.20: absolute error distribution of the model predictions with regards to the labels for higher terms of mode III

As shown in Fig. 4.19, the training converges at an error level of around $5 \cdot 10^{-2}$, but after the symbolic recovery the performance of the model drops. This is further illustrated in the error plot Fig. 4.20, which is high compared to a successful training, e.g Fig. 4.12. The model architecture after symbolic fixing is shown in Fig. 4.21. Only two edges remain in the first layer after pruning. The symbolically recovered formula is

$$f(r, \phi) \approx 6.46 - 0.06 (-3.2r - 9.8)^2 (0.14 \sin(5.5\phi - 0.009)). \quad (4.9)$$

The expression indicates that the model was unable to recover the formula successfully, as it deviates greatly from the underlying function 4.8. This behaviour was consistent over multiple runs with comparable hyperparameter settings.

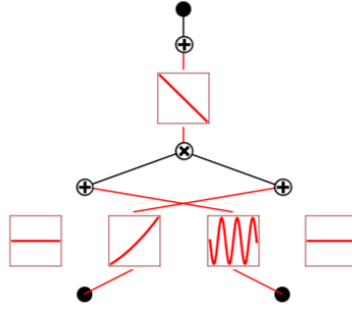


Figure 4.21: model plot for higher terms of displacement fields for mode III after symbolic fixing

4.5 PARETO FRONT FOR DIFFERENT COMPLEXITY LIBRARIES

To analyze the trade-off between model complexity and performance, Pareto fronts were visualized. For this purpose 350 models were trained for different combinations of KAN architectures with two layers and three layers. The maximum number of addition n^a and multiplication nodes n^m were set to 5 respectively:

$$n^a \in [0, 1, 2, 3, 4, 5], \quad n^m \in [0, 1, 2, 3, 4, 5], \quad (4.10)$$

which results in $\text{kan_width} = [2, [n^a, n^m], [n^a, n^m], 1]$, $\text{depth} = 3$.

For computational reasons, not all permutations of n^a and n^m were evaluated for $\text{depth} = 3$, so that layer 1 and layer 2 had the same number of respective nodes. Additionally, because Liu et al.[15] state that the initial grid is important for training success, models were trained with initial gridsizes from $G = 3$, to $G = 50$. The function used to generate the data was:

$$f(r, \phi) = 3\sqrt{r} \sin \frac{\phi}{2}. \quad (4.11)$$

Since the assignment of the complexity values in the pykan library is not explicitly described in the literature, it is compared to a library in which all analytical functions are assigned the same value $c_{lib} = 1$.

4.5.1 Library with Weighted Complexity

The library in the pykan package assigns complexity values c_{lib} to analytical function. The training of models with different initialization grids and architectures is investigated.

Of the 350 models that were initialized, 151 successfully recovered a symbolic expression. The most successful models can be seen in Tab 4.5. “Reduced expression” denotes that the originally recovered function was reduced to an expression containing only analytical functions defined in the library.

ID	RMSE	kan_width	Grid	c_{kan}	reduced expression	c_{func}
242	$1.27 \cdot 10^{-7}$	[[2,0],[5,3],[1,0]]	[20,50,100,200]	5	$\sqrt{r} \sin(\phi)$	4
173	$1.38 \cdot 10^{-7}$	[[2,0],[5,4],[1,0]]	[10,20,50,100,200]	5	$\sqrt{r} \cos(\phi)$	4
212	$1.38 \cdot 10^{-7}$	[[2,0],[0,3],[1,0]]	[20,50,100,200]	5	$\sqrt{r} \sin(\phi)$	4
162	$1.39 \cdot 10^{-7}$	[[2,0],[3,5],[1,0]]	[10,20,50,100,200]	5	$\sqrt{r} \cos(\phi)$	4
100	$1.42 \cdot 10^{-7}$	[[2,0],[5,1],[1,0]]	[5,10,20,50,100,200]	5	$\sqrt{r} \sin(\phi)$	4
295	$1.46 \cdot 10^{-7}$	[[2,0],[2,4],[1,0]]	[50,100,200]	5	$\sqrt{r} \sin(\phi)$	4
329	$1.52 \cdot 10^{-7}$	[[2,0],[2,3],[2,3],[1,0]]	[50,100,200]	6	$\sqrt{r} \cos(\phi)$	4

Table 4.5: The 7 most successful models for the weighted library, according to RMSE. c_{kan} is the number of active edges after symbolic fixing the model, c_{func} the complexity of the expression with respect to the original pykan library.

The Pareto fronts are plotted for both the complexity regarding the number of edges used c_{kan} (see Fig. 4.22), as well as the complexity of the reduced expression c_{func} (see Fig. 4.23). In Fig. 4.23, we also highlighted 12 expressions that match the composition of the underlying function (square root of r times trigonometric function). As a final result, 12 out of 350 models were able to recover the underlying function successfully.

4.5.2 Library with Constant Complexity

The alternative library assigns each analytical function the value $c_{lib} = 1$, which reduces the selection criterion to the R^2 score alone. The Pareto fronts show both the structural complexity c_{kan} used in the models (see Fig. 4.24), as well as the complexity of the reduced expression c_{func} (see Fig. 4.25). Table 4.6 lists the most successful models in terms of RMSE. Both the calculated complexities and the length of the recovered expression indicate that the models were less successful than those obtained with the weighted pykan library. Overall, 101 out of 350 initialized models were able to recover a symbolic expression.

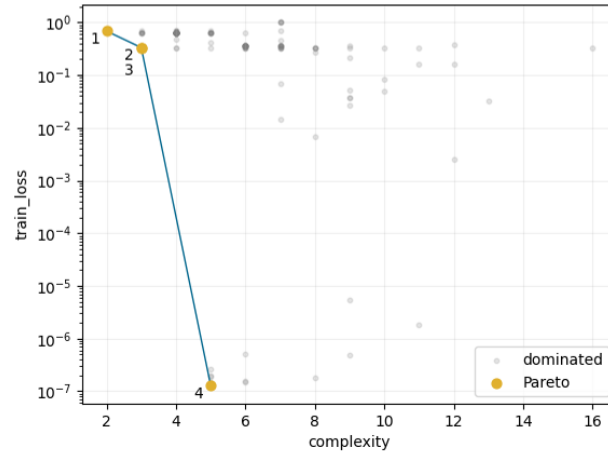


Figure 4.22: Pareto front for weighted complexity terms c_{kan} . “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow.

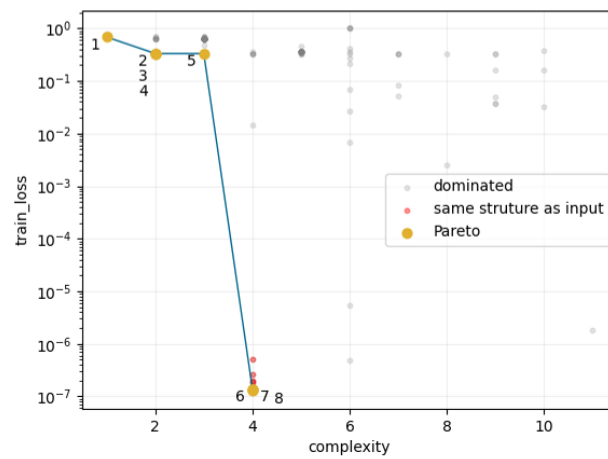


Figure 4.23: Pareto front for weighted complexity c_{func} for the reduced expressions. “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow. Models that resulted in the same expression structure as the underlying function for data generation are marked red.

ID	RMSE	kan_width	Grid	c_{kan}	reduced expression	c_{func}
0	$1.56 \cdot 10^{-4}$	[[2,0],[0,2],[1,0]]	[5,10,20,50,100,200]	3	$\arccos(\sqrt{r} \sin(\phi))$	4
1	$4.28 \cdot 10^{-4}$	[[2,0],[0,2],[0,2],[1,0]]	[50,100,200]	3	$\tan(\sqrt{r} \sin(\phi))$	7
2	$5.49 \cdot 10^{-3}$	[[2,0],[2,1],[2,1],[1,0]]	[5,10,20,50,100,200]	8	$\operatorname{atanh}(\exp(\dots))$	17
3	$6.64 \cdot 10^{-3}$	[[2,0],[1,5],[1,5],[1,0]]	[3,5,10,20,50,100,200]	9	$\arccos(\dots)$	22
4	$1.44 \cdot 10^{-2}$	[[2,0],[5,5],[5,5],[1,0]]	[5,10,20,50,100,200]	5	$\tan(\dots \sin(\dots))$	13
5	$2.40 \cdot 10^{-2}$	[[2,0],[3,2],[3,2],[1,0]]	[50,100,200]	10	$\exp(\dots)$	33
6	$2.95 \cdot 10^{-2}$	[[2,0],[3,0],[1,0]]	[3,5,10,20,50,100,200]	5	$\sin(\operatorname{atan}(\dots))$	9

Table 4.6: The 7 most successful models for the constant library, according to RMSE. c_{kan} is the number of active edges after symbolic fixing the model, c_{func} the complexity of the expression with regards to the original pykan library. Because the length of some expressions exceeds the available space they are only hinted at.

Fig. 4.24 shows the Pareto front for c_{kan} , the number of edges used during symbolic recovery. The most successful models had three unpruned edges after symbolic fixing. However, the RMSE is higher than that achieved with the weighted complexity values c_{lib} used in the pykan library by approximately three orders of magnitude. Additionally, both Fig. 4.25 and Table 4.6 indicate that no experiment was able to recover a symbolic formula with the same structure as the function used to generate the synthetic data set. Accordingly, none of the 350 models recovered the underlying formula structure successfully.

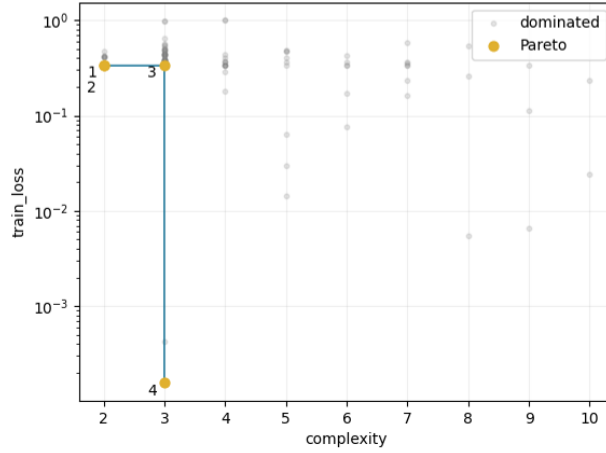


Figure 4.24: Pareto front for constant complexity values c_{kan} . “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow.

4.6 SYNTHETIC WILLIAMS FIELD DATA

Data provided by the DLR describes a simulated displacement field u around a crack tip. As an exploratory step towards predicting displacement fields from simulated experimental data, a model is trained on the u_x component of the displacement field. The simulated data is presented in Fig. 4.26.

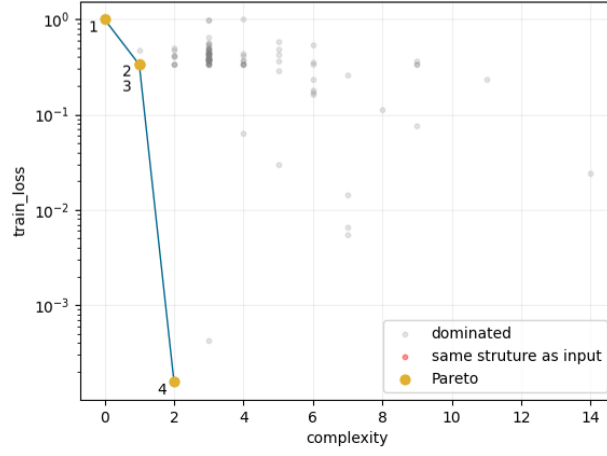


Figure 4.25: Pareto front for constant complexity values c_{func} for the reduced expressions. “train_loss” denotes the RMSE, dominated experiments are displayed in grey, models that are part of the Pareto front are shown in yellow. Models that resulted in the same expression structure as the underlying function for data generation are marked red.

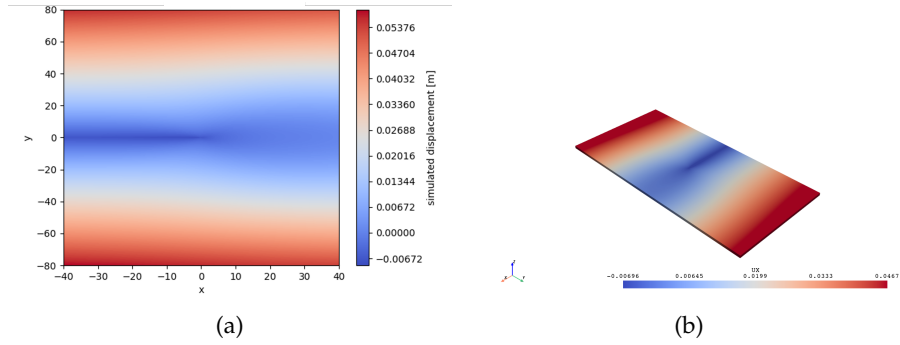


Figure 4.26: Distribution of simulated data for the displacement field u_x . (a) shows the 2D distribution in the x - y plane, while (b) shows the corresponding 3D representation

The model's prediction in Fig. 4.27a reveals that the outline of the crack tip is recognizable. Apart from this, the prediction does not accurately reproduce the displacement field. The absolute error of the prediction with respect to the provided labels is of the same order of magnitude as the values for the displacement itself (see Fig. 4.27b).

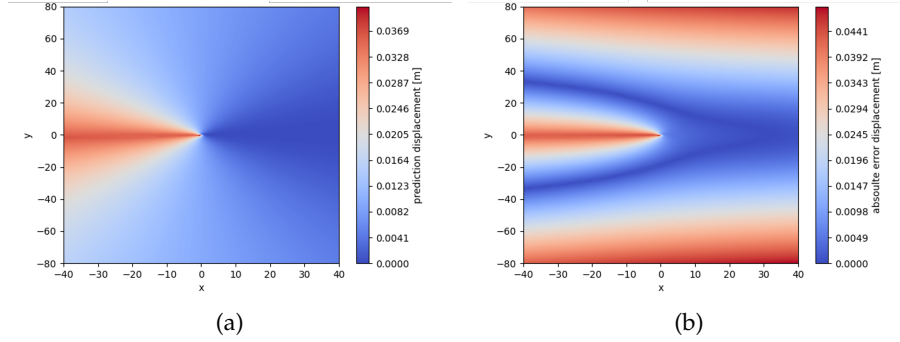


Figure 4.27: (a) shows the prediction of the displacement field u_x by a KAN, while (b) shows the absolute error of the prediction for displacement field u_x w.r.t labels

In Fig. 4.28 we see the final model architecture for the displacement field. The KAN did not identify trigonometric dependencies in the data. Additionally successful training was only achieved for a single grid, with grid extensions resulting in numerical error (NaN) outputs. Increasing depth and width of the KAN architecture failed to deliver results as well. The same behaviour was observed for displacement field u_y and u_z .

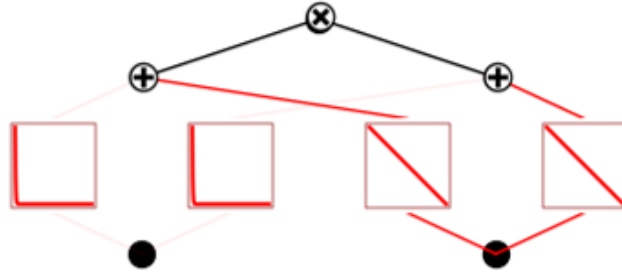


Figure 4.28: model architecture for KAN model trained on simulated data for displacement field u_x .

DISCUSSION

5.1 SYMBOLIFICATION METHOD FOR EDGES

In Sec. 4.1.1 the differences between automatic and manual symbolification were analyzed. Given prior knowledge, using manual symbolification is preferable, as we can guide the model to apply certain analytic functions on specific edges for symbolic recovery. However, in most cases we either do not have prior knowledge or assume we do not have it. Therefore, the option of fixing edges manually to specific functions is disregarded.

On the other hand, the example experiment shows that automatic symbolification presents considerable limitations. The selection criterion Eq. 3.7 serves as basis for the selection and operates locally on each edge. As a result, the global structure of the underlying function is not taken into account, which can lead to suboptimal representations for the entire network. In Fig. 4.2 it is apparent that the initial RMSE during the symbolic recovery was higher for the manually fixed model. However, during training the manual symbolification performed better in terms of RMSE, and achieved a lower RMSE. A possible method for improvements to the selection process of the auto_symbolic functions is adjusting either the selection criterion or introducing a library with adjusted complexities c_{lib} . The impact of adjusting the selection criterion is further discussed in Sec. 5.4.

5.2 BEHAVIOUR NEAR SINGULARITIES

In Sec. 4.1.2 the results for behaviour near singularities were presented. Looking at Fig. 4.8, it proves to be difficult to successfully model functions that have exploding gradients. In addition to the presented results, many similar experiments failed to produce numerical values for gradient and parameters. Experiments that produced results close to a singularity exhibited the same issues as presented in Sec. 4.1.2. This behaviour is expected, since we try to approximate the underlying function with B-splines. For smaller grid intervals near singularities, the gradient becomes too steep for B-spline approximation, and the training fails to converge, resulting in NaN values, or alternatively, infinite values. Such behaviour leads to invalid gradients, which prevent updates of the model parameters. Ultimately, the training process is often terminated, or the model fails to produce meaningful results. Consequently, we disregarded the stress fields for this experiment, and focused on displacement fields, in particular on the displacement field of mode III (see Sec. 2.4).

5.3 HYPERPARAMETER OPTIMIZATION

The hyperparameter optimization proved to be challenging. Different KAN architectures need specific hyperparameter configuration to perform well. In an Optuna study that adjusts both the architecture as well as the hyperparameter, the search space prohibitively large, so that results may be not consistent enough to make confident statements about the different hyperparameters. On the other hand, assuming a specific architecture for specific problems requires prior knowledge which is typically not available when searching for formulas that have not been discovered yet. This trade-off becomes apparent in the first Optuna study A (see Sec. 4.3). Many trials result in an objective value that evidently did not result in successful symbolification of the underlying function. This result is valuable, because it highlights the difficulty of finding optimal configurations of the search space. However, this reduces the trials we want to examine closer to a small subset. The time required to carry out a study limits the achievable number of trial. Having said that, the result of the study, that λ_{entropy} and λ_{L1} are the most influential regularization parameters is consistent with both our observations as well as the emphasis placed on those hyperparameters in literature [15] [4].

In study B similar results can be observed. Again λ_{entropy} and λ_{L1} are the most influential hyperparameters. The KAN architecture was successfully pruned during training. Still, the symbolically recovered expression for the best trial (see B.1) was not able to achieve the desired structure. Both study A and study B illustrate the importance of λ_{entropy} and λ_{L1} , but fail to provide hyperparameter values that generalize well enough to be used going forward. This suggests that not initializing a more optimized architecture for the task limited the effect of the hyperparameters. Also the search space we presented to the optimization study could have been too narrow or too broad.

Study C provides evidence that an architecture which suits the problem at hand can lead to great results. When looking at the distribution of the different depths in Fig. 4.17, a clear pattern is observable. KANs with architecture `kan_width = [2, [0, 1]]` seem to be able to approximate the underlying function effectively. However, the deeper the model the worse the performance appears to be, which again could be attributed to an increasing model complexity. We will investigate this claim in Sec. 5.4. Node threshold also seems to be of secondary importance compared to depth. The importance of `width_add` can be explained by looking at the way Optuna optimizes. The user specifies which parameters are to be optimized, and for each trial the optimizer chooses a value for each parameter. When `depth = 1` is chosen, `width_add` does not have any influence over the KAN architecture anymore. The reason is that two input nodes are given, and an output node `[0,1]`.

Depth = 1 leads to a KAN architecture of $\text{kan_width} = [2, [0, 1]]$, regardless of which other parameters were chosen for width_add or width_mult . This introduces a bias into the optimization process, as specific values for hyperparameters render other hyperparameters obsolete because of the resulting KAN architecture. All studies highlight that hyperparameter optimization, as conducted in this master thesis, is insufficient without stronger constraints on model architecture. There introduces a trade-off between initial model architecture and limiting the impact of prior knowledge in the setup.

5.3.1 Higher Term Exploration

As a result of the hyperparameter optimization the standard parameters of the pykan-package were used for the evaluation of multiple terms of the Williams series for the displacement field in mode III. However, the assumption that we do not know the optimal architecture for the underlying formula is still present. As such, the KAN architecture $\text{kan_width} = [2, [5, 5], 1]$ was chosen.

In Fig. 4.21 we see a sparse KAN model with only two active edges. Three edges per input parameter would have been expected, because we generated a dataset with the underlying function for the first three William terms for mode III of the displacement field. The symbolification for the edge connected to input ϕ is an oscillating function. It may represent superpositions of multiple trigonometric functions. This indicates that too many edges were pruned during training, likely due to overly strong regularization. In Fig. 4.19 we can see that after symbolification the model does not achieve a better RMSE, and the model was too sparse which led to underfitting. As shown in 5.3 the hyperparameters prove to be highly sensitive. However, for functions with multiple additive terms, the results of study B and the exploration of higher terms in the William Series seem to suggest that the standard regularization parameter values regularize too strongly. Further exploration is needed to confirm this observation. The results also indicates that the optimal parameter configuration depends on the underlying function of the dataset.

5.4 PARETO ANALYSIS

The Pareto analysis was conducted with hyperparameters from Sec. 4.2, where the underlying formula of the dataset was successfully recovered. The formula is given by

$$f(r, \phi) = 3\sqrt{r} \sin\left(\frac{\phi}{2}\right). \quad (5.1)$$

One motivating factor for conducting the Pareto study is to investigate the influence of the initial grid size, the other is to investigate the importance of

the complexity term c_{lib} .

For the pykan-library 12 out of 350 models were able to recover the correct symbolic structure of the underlying function $f(r, \phi)$. Furthermore, 151 out of 350 models were able to recover a symbolic expression (see Sec. 4.5). Conversely, this implies that nearly $\frac{4}{7}$ of experiments done with the pykan-library were abandoned in training. This points to the importance of grid initialization for successful training, as is stated by Liu et al. [15]. Interestingly, the experiment with almost identical parameters to those in Sec. 4.2 was also unable to complete training and was terminated during the final training after symbolic recovery. The only difference was the seed given in the model's initialization.

The fact that only 12 out of 350 models were able to recover the correct underlying structure even under favourable conditions highlights the sensitivity of KANs to choice of hyperparameters. The instability in the training process is further indicated by the difference in results of experiments where the only distinguishing factor was the seed.

The custom library, where all complexity terms c_{lib} were 1, did not result in any symbolification that managed to match the structure of the underlying function. This behaviour signifies the importance of the complexity term c_{lib} , which acts as structural regularization for the selection. Without it, the selection process reduces to function approximation, where multiple functions may achieve comparable R^2 scores. This proves to be insufficient, as it prevents reliable symbolic recovery. Additionally, only 101 out of 350 attempted models succeeded in finding a symbolic expression for the custom library, which indicates a worse performance in general. Therefore, it can be concluded that assigning complexity c_{lib} to each analytic function in the library is crucial, given the significantly worse performance of the Pareto analysis with the alternative library.

5.5 SYNTHETIC WILLIAMS FIELD DATA

The experiment on the synthetic Williams field data in Sec. 4.6 was not successful. The only KAN architecture that produced viable results was `kan_width = [2, [0, 1]]`, which in theory could recover the first term of the Williams Series, but it did not. A problem was the inability to successfully perform a grid refinement, which could indicate that the default configuration is insufficient for complex superpositions.

The purpose of this experiment was to investigate whether a standard KAN without any hyperparameter optimization could recover parts of the Williams series as all three modes contribute to the displacement field u_x . This was not achieved, which is to be expected, due to the previously observed sensitivity of KANs.

CONCLUSION AND OUTLOOK

6.1 CONCLUSION

This thesis explored whether Kolmogorov-Arnold networks can be used to recover components from the Williams series. Furthermore, it investigated whether KANs are capable of successful symbolification without adapting the architecture across several initializations.

The main finding of this work was that Kolmogorov-Arnold networks are capable of recovering underlying formulas from data. The first term of the displacement field in mode III of the Williams series could be recovered successfully. However, the recovery of higher-order terms was not successful.

A central challenge throughout all experiments was the strong sensitivity of KANs to hyperparameters. The performance of the models was critically dependent on many hyperparameters, and they had to be adjusted for each specific experiment.

Hyperparameter optimizations and Pareto front analysis did not bring the desired results and were themselves dependent on the choice of hyperparameter values. This further highlights the difficulty of identifying suitable configurations.

The network architecture proved to be another limitation. KANs are not able to prune complete layers. This leads to more complex models, can hinder successful training and degrades performance, if more layers than necessary were initialized. However, wider and deeper networks are preferable, since we cannot make claims over the data's underlying functional structure if we do not possess prior knowledge.

Combined, this makes symbolic recovery for unknown relations difficult, because for optimal results a certain degree of prior knowledge seems necessary, which is not possible for investigating new relationships in data. Additionally, even if an experiment was carefully initialized and trained using deliberately chosen hyperparameters, the success of the training can still be dependent on the seed. Using a predefined seed is not desirable, because for unknown relations the optimal seed, cannot be known as well. This contributes to the difficulty of using KANs for discovering new scientific relations.

The approach used in this thesis was utilizing single wide and deep KANs for recovering formulas. Contrary to the results presented in this work, existing approaches, such as Buhler and Guillén-Gosálbez [4], use algorithms across multiple models to reduce the KAN architecture.

Their approach does not have the problem of too complex networks, since they built KANs from $\text{depth} = 1$ and $\text{width} = 1$ up, stopping at a desired result with minimal architecture. That could be a reason why the results in

their applications were more promising than the results presented here.

In summary, KANs display the ability to recover the first term of the displacement field for mode III of the Williams series and other simpler formulas well. However, due to their sensitivity to hyperparameters and their rigid architecture, they are currently limited in their effectiveness for symbolic regression, in the applications shown in this thesis.

6.2 OUTLOOK

To improve the performance of my models, a more systematic thorough hyperparameter optimization is necessary. In particular, it is important to enable more robust training environments that are not as sensitive to change. The implied connection of weaker regularisation and increased performance for higher terms should be investigated. Future studies could also explore more nuanced implementations of the library complexity c_{lib} .

For symbolic regression tasks specifically, a way for implementing a global component in symbolic recovery could be explored, that bridges the dependency on purely local symbolification.

Furthermore, a comprehensive comparison to other implementation using KANs for symbolic recovery should be done, and complimentary approaches developed. Other published work uses different trainable functions instead of splines. This is an avenue that is worth exploring.

For KANs to be used as a continuous model in real-world symbolic regression applications, the focus should be put on enabling more robust models, and more flexible architectures.

Part II

APPENDIX

LIBRARIES PARETO FRONT

For both libraries, the dictionary was adjusted to display only the complexity c_{lib} value.

A.1 LIBRARY WITH PREDEFINED COMPLEXITY

```
SYMBOLIC_LIB = {'x': 1,,
                'x^2': 2,
                'x^3': 3,
                'x^4': 3,
                'x^5': 3,
                '1/x': 2 ,
                '1/x^2': 2,
                '1/x^3': 3,
                '1/x^4': 4,
                '1/x^5': 5,
                'sqrt': 2,
                'x^0.5': 2,
                'x^1.5': 4,
                '1/sqrt(x)': 2,
                '1/x^0.5': 2,
                'exp': 2,
                'log': 2,
                'abs': 3,
                'sin': 2,
                'cos': 2,
                'tan': 3,
                'tanh': 3,
                'sgn': 3,
                'arcsin': 4,
                'arccos': 4,
                'arctan': 4,
                'arctanh': 4,
                '0': 2,
                'gaussian': 3,
                }
```

A.2 LIBRARY WITH CONSTANT COMPLEXITY

```
SYMBOLIC_LIB = {'x': 1,
                'x^2': 1,
```

```
'x^3': 1,  
'x^4': 1,  
'x^5': 1,  
'1/x': 1,  
'1/x^2': 1,  
'1/x^3': 1,  
'1/x^4': 1,  
'1/x^5': 1,  
'sqrt': 1,  
'x^0.5': 1 ,  
'x^1.5': 1,  
'1/sqrt(x)': 1,  
'1/x^0.5': 1,  
'exp': 1,  
'log': 1,  
'abs': 1,  
'sin': 1,  
'cos': 1,  
'tan': 1,  
'tanh': 1,  
'sgn': 1,  
'arcsin': 1,  
'arccos': 1,  
'arctan': 1,  
'arctanh': 1,  
'0': 1,  
'gaussian': 1,  
  
}
```

RECOVERED FORMULA OPTUNA STUDY

B.1 RECOVERED FORMULA OPTUNA STUDY

For Study B (see [4.3.2](#)) the recovered formula of the best trial is

$$-0.3\phi - (4.2\sqrt{1.5 * r + 7.6} - 11.88)(-0.64 \sin(3.6\phi + 1.7) - 0.0084) - 0.017$$

BIBLIOGRAPHY

- [1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. “Optuna: A Next-generation Hyperparameter Optimization Framework.” In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.
- [2] David Bertoin, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. “Numerical influence of ReLU’(o) on backpropagation.” In: (2021). Version Number: 4. DOI: [10.48550/ARXIV.2106.12915](https://doi.org/10.48550/ARXIV.2106.12915).
- [3] Andriy Burkov. *Machine learning engineering*. Quebec City, Canada: True Positive Inc, 2020. 282 pp. ISBN: 978-1-9995795-7-9.
- [4] Marco Andrea Bühler and Gonzalo Guillén-Gosálbez. *KAN-SR: A Kolmogorov-Arnold Network Guided Symbolic Regression Framework*. 2025. DOI: [10.48550/ARXIV.2509.10089](https://doi.org/10.48550/ARXIV.2509.10089).
- [5] Renáta Dubčáková. “Eureqa: software review.” In: *Genetic Programming and Evolvable Machines* 12.2 (June 2011), pp. 173–178. DOI: [10.1007/s10710-010-9124-z](https://doi.org/10.1007/s10710-010-9124-z).
- [6] Paul H. C. Eilers and Brian D. Marx. “Flexible smoothing with B-splines and penalties.” In: *Statistical Science* 11.2 (May 1, 1996). DOI: [10.1214/ss/1038425655](https://doi.org/10.1214/ss/1038425655).
- [7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning. Adaptive computation and machine learning*. Cambridge, Massachusetts: The MIT Press, 2016. 775 pp. ISBN: 978-0-262-03561-3.
- [8] Qian Yue Hao, Fengli Xu, Yong Li, and James Evans. “Artificial intelligence tools expand scientists’ impact but contract science’s focus.” In: *Nature* 8099 (Jan. 29, 2026), pp. 1237–1243. DOI: [10.1038/s41586-025-09922-y](https://doi.org/10.1038/s41586-025-09922-y).
- [9] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, eds. *An introduction to statistical learning: with applications in R*. Springer texts in statistics 103. New York: Springer, 2013. 426 pp. ISBN: 978-1-4614-7137-0.
- [10] Johannes Kepler. *Astronomia nova*. Noscemus Wiki, last revision: 03.02.2026. URL: http://wiki.uibk.ac.at/noscemus/Astronomia_nova (visited on 05/17/2026).
- [11] John R. Koza. “Genetic programming as a means for programming computers by natural selection.” In: *Statistics and Computing* 4.2 (June 1994). ISSN: 0960-3174, 1573-1375. DOI: [10.1007/BF00175355](https://doi.org/10.1007/BF00175355).

- [12] Rudolf Kruse, Sanaz Mostaghim, Christian Borgelt, Christian Braune, and Matthias Steinbrecher. *Computational Intelligence: A Methodological Introduction*. Texts in Computer Science. Cham: Springer International Publishing, 2022. ISBN: 978-3-030-42226-4. DOI: [10.1007/978-3-030-42227-1](#).
- [13] Meinhard Kuna. *Finite Elements in Fracture Mechanics: Theory - Numerics - Applications*. Vol. 201. Solid Mechanics and Its Applications. Dordrecht: Springer Netherlands, 2013. ISBN: 978-94-007-6679-2. DOI: [10.1007/978-94-007-6680-8](#).
- [14] Ziming Liu, Max Tegmark, Pingchuan Ma, Wojciech Matusik, and Yixuan Wang. "Kolmogorov-Arnold Networks Meet Science." In: *Physical Review X* 15.4 (Dec. 17, 2025), p. 041051. ISSN: 2160-3308. DOI: [10.1103/4t7t-v19l](#).
- [15] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. *KAN: Kolmogorov-Arnold Networks*. 2024. DOI: [10.48550/ARXIV.2404.19756](#).
- [16] Nour Makke and Sanjay Chawla. "Interpretable scientific discovery with symbolic regression: a review." In: *Artificial Intelligence Review* 57.1 (Jan. 2024). DOI: [10.1007/s10462-023-10622-0](#).
- [17] Lucie Malíková and Václav Veselý. "Williams Expansion Terms and their Importance for Accurate Stress Field Description in Specimens with a Crack." In: *Transactions of the VŠB - Technical University of Ostrava, Mechanical Series* 59.2 (Dec. 30, 2013), pp. 109–114. ISSN: 12100471, 18040993. DOI: [10.22223/tr.2013-2/1963](#).
- [18] David Melching and Eric Breitbarth. "Advanced crack tip field characterization using conjugate work integrals." In: *International Journal of Fatigue* 169 (Apr. 2023), p. 107501. ISSN: 01421123. DOI: [10.1016/j.ijfatigue.2023.107501](#).
- [19] David Melching, Tobias Strohmann, Guillermo Requena, and Eric Breitbarth. "Explainable machine learning for precise fatigue crack tip detection." In: *Scientific Reports* 1 (June 9, 2022), p. 9513. DOI: [10.1038/s41598-022-13275-1](#).
- [20] Marius-Constantin Popescu, Valentina Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. "Multilayer perceptron and neural networks." In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009). URL: http://www.researchgate.net/publication/228340819_Multilayer_perceptron_and_neural_networks.
- [21] Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Birmingham, UK: Packt Publishing, 2022. ISBN: 978-1801819312.
- [22] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead." In: (2018). DOI: [10.48550/ARXIV.1811.10154](#).

- [23] Michael Schmidt and Hod Lipson. "Distilling Free-Form Natural Laws from Experimental Data." In: *Science* 324 (Apr. 3, 2009). DOI: [10.1126/science.1165893](https://doi.org/10.1126/science.1165893).
- [24] Guido F. Smits and Mark Kotanchek. "Pareto-Front Exploitation in Symbolic Regression." In: *Genetic Programming Theory and Practice II*. Ed. by Una-May O'Reilly, Tina Yu, Rick Riolo, and Bill Worzel. Vol. 8. New York: Springer-Verlag, 2005, pp. 283–299. ISBN: 978-0-387-23253-9. DOI: [10.1007/0-387-23254-0_17](https://doi.org/10.1007/0-387-23254-0_17).
- [25] L.V. Stepanova and O.N. Belova. "Coefficients of the williams power expansion of the near crack tip stress field in continuum linear elastic fracture mechanics at the nanoscale." In: *Theoretical and Applied Fracture Mechanics* 119 (June 2022). ISSN: 01678442. DOI: [10.1016/j.tafmec.2022.103298](https://doi.org/10.1016/j.tafmec.2022.103298).
- [26] *The History of Johannes Kepler*. NASA Jet Propulsion Laboratory (JPL). URL: <https://www.jpl.nasa.gov/news/the-history-of-johannes-kepler/> (visited on 05/17/2026).
- [27] Silviu-Marian Udrescu and Max Tegmark. "AI Feynman: A physics-inspired method for symbolic regression." In: *Science Advances* 6.16 (Apr. 17, 2020). ISSN: 2375-2548. DOI: [10.1126/sciadv.aay2631](https://doi.org/10.1126/sciadv.aay2631).
- [28] M. L. Williams. "On the Stress Distribution at the Base of a Stationary Crack." In: *Journal of Applied Mechanics* 24.1 (Mar. 1, 1957), pp. 109–114. ISSN: 0021-8936, 1528-9036. DOI: [10.1115/1.4011454](https://doi.org/10.1115/1.4011454).
- [29] Runpeng Yu, Weihao Yu, and Xinchao Wang. *KAN or MLP: A Fairer Comparison*. 2024. DOI: [10.48550/ARXIV.2407.16674](https://doi.org/10.48550/ARXIV.2407.16674).
- [30] Zarita Zainuddin and Ong Pauline. "Function approximation using artificial neural networks." In: *WSEAS Transactions on Mathematics* 7.6 (2008), pp. 333–338. URL: https://www.researchgate.net/publication/228840414_Function_approximation_using_artificial_neural_networks.